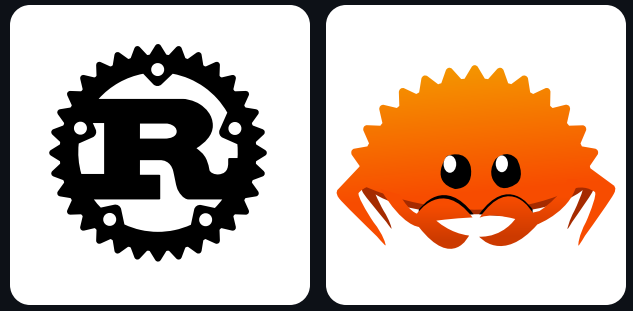
Introduction to the Rust programming Language



Following along The Rust Book from the official source

by: **Simon Lalonde**

For: **IFT-769** (Theoritical concepts CS)

📖 **Project overview -** Going through "The Rust Programming Language"

The Rust Programming Language by Steve Klabnik and Carol Nichols

Book overview:

- Official guide to the Rust programming language
- Covers the basics (syntax, types, functions)
- Build tool and package manager (Cargo)
- Advanced and Rust-specific features:
  - Ownership, borrowing, lifetimes
  - Unique error handling
  - Concurrency

Klabnik, Steve, and Carol Nichols. The Rust Programming Language. 2nd ed., No Starch Press.

📚 **Theoretical concepts** - Key topics covered

1. Common Programming Concepts (*variables, types, control flow*)
2. Understanding Ownership (*memory management*)
3. Structs, Enums and Pattern Matching
4. Containers/Collections
5. Error Handling
6. Generics, Traits and Lifetimes
7. Functional and OO features
8. Smart pointers and Concurrency
9. Patterns and matching + Advanced features

Klabnik, Steve, and Carol Nichols. The Rust Programming Language. 2nd ed., No Starch Press.

🛠️ **Practical project #1 -** Write an I/O CLI program

Halfway project for a `grep` clone CLI app covers:

1. Code organization (crates, modules)
2. Use of containers and strings
3. Error handling
4. Using traits and lifetimes
5. Testing and documentation

Klabnik, Steve, and Carol Nichols. The Rust Programming Language. 2nd ed., No Starch Press.

🛠️ **Practical project #2 -** Building a Multithreaded Web Server

Final Project from the book includes :

1. Learn TCP/IP networking and HTTP
2. Listen to TCP connections on a socket
3. Parse HTTP requests
4. Generate HTTP responses
5. Handle multiple requests concurrently with a thread pool

Klabnik, Steve, and Carol Nichols. The Rust Programming Language. 2nd ed., No Starch Press.

Rust Overview

- **Systems programming language** focused on **safety** and **performance**
- TODO

Currently known projects

TODO

Predicted use cases

TODO

Pros and Cons of Rust

PROS:

- **Memory safety**: No null pointers, dangling pointers, or buffer overflows
- **Error handling**: With the `Result` and `Option` types
- **Concurrency**: Safe and efficient with the ownership system
- **Performance**: Comparable to C/C++ with zero-cost abstractions
- **Ecosystem**: Growing with a strong community and package manager (**Cargo**)
- **Helpful compiler**: Provides detailed error messages and warnings

CONS:

- **Learning curve**: Ownership, borrowing, and lifetimes can be challenging
- **Tooling and prevalence**: Not as mature as other languages (C/C++, Python, etc.)
- **Syntax**: Can be verbose and complex compared to other languages

👩‍💻 Installation and setup

Installation:

1. Install Rust using `rustup` (Rust toolchain installer)

Included toolchain:

- `rustc`: Rust compiler
- `rustup`: Rust toolchain manager
- `rustfmt`: Rust code formatter
- `cargo`: Rust package manager and build tool

Package and library management

- **Crates** are Rust packages that can be shared and reused
- Managed with **Cargo**, the Rust package manager

👨‍💻 **Setup example** - Hello World! (1/2)

Env setup and features:

- Easy install: `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
- Rustup for managing toolchains: `rustup update`
- Included formatter: `rustfmt --check src/main.rs` (dry-run mode)
- Cargo for building and managing projects: `cargo new project_name`
- Quality of life with `rust-analyzer`: LSP, build/debug IDE support etc.

👨‍💻 **Building with cargo** - Hello World! (2/2)

To **initialize** a new project use `cargo new hello_world`. The **structure** will include a `src/` dir for code, `Cargo.toml` config file, `Cargo.lock` for dependencies and version and `target/` for build artifacts:

Useful Cargo commands when building a project:

- `cargo build` or `cargo run` to compile and run the project. Use `--release` flag for compilation with optimizations inside `target/release/`
- `cargo check`: Check the project for errors without building
- `cargo doc`: Generate documentation for the project
- `cargo clean`: Remove build artifacts
- `cargo update`: Update dependencies
- `cargo fmt`: Format the code according to the Rust style guidelines
- `cargo test`: Run tests in the project

THEORY STUFF

TODO

⌨️ **Guessing Game Project** - Basic concepts (1/2)

TODO

⌨️ **Guessing Game Project** - Crates features (1/2)

- Using another crate: `rand` for random number generation
- `cargo doc --open` to generate and open documentation

-etc.