

Documentation window

September 1, 2025 → September 4, 2025

1) Title & Objective

Project Title: Rust Calculator – A Fun, Stylish Rust CLI Tool

Objective:

Document a guided, prompt-powered learning journey into Rust (from Python) on Ubuntu 24.04 LTS, culminating in a working CLI calculator that greets the user, remembers their name, and supports basic arithmetic.

2) Quick Summary of the Technology

Rust is a modern systems programming language focused on performance and memory safety, without using a garbage collector.

Where is it used?

- High-performance backends and services
- CLI tools
- Embedded systems and OS-level development
- WebAssembly

Real-world examples: Mozilla, Dropbox, Cloudflare, and others.

3) System Requirements

OS: Ubuntu 24.04 LTS (Linux), macOS, or Windows

Tools:

- Rust toolchain (rustup + cargo)
- VS Code with Rust Analyzer extension

Libraries: Standard library only (std::io)

4) Installation & Setup Instructions

```
$ cd path/to/folder
$ cargo new rust_calculator
$ cd rust_calculator
$ cargo build
$ cargo run
```

5) Minimal Working Example

```
use std::io;

fn main() {
    println!("Hello from Davids First Rust Calculator!");
    println!("I was created on September 3rd 2025");

    println!("Whats Your Name?");
    let mut user_name = String::new();
    io::stdin().read_line(&mut user_name).expect("Failed to read line");
    println!("Hello {}! Lets do some Maths!", user_name);
}
```

```

let mut input_a = String::new();
println!("Please enter the first number:");
io::stdin().read_line(&mut input_a).expect("Failed to read line");

let mut input_op = String::new();
println!("Please Enter an operator (+, -, *, /):");
io::stdin().read_line(&mut input_op).expect("Failed to read line");

let mut input_b = String::new();
println!("Please enter the second number:");
io::stdin().read_line(&mut input_b).expect("Failed to read line");

let a: f64 = input_a.trim().parse().expect("Please type a number!");
let op = input_op.trim();
let b: f64 = input_b.trim().parse().expect("Please type a number!");

match op {
    "+" => println!("{}", user_name, a + b),
    "-" => println!("{}", user_name, a - b),
    "*" => println!("{}", user_name, a * b),
    "/" => {
        if b != 0.0 {
            println!("{}", user_name, a / b);
        } else {
            println!("{}", user_name, "there is an Error: You can't divide by zero!");
        }
    }
    _ => println!("{}", user_name, "That's an Invalid operator"),
}
}

```

6) AI Prompt Journal

A) Step 1 — Creating a Learning Journey Plan Prompt

Link:

https://ai.moringaschool.com/ai-software/learning-with-ai/learning-new-language/#_creating_a_learning_journey_plan

Prompt Used:

Mastering Rust Coming from Python. Using an Ubuntu 24.04 LTS OS.

****End Goal: Develop proficiency in Rust for Build a simple yet impressive Rust CLI Tool****

Phase 1: Language Fundamentals

- Prerequisite: Basic programming knowledge in Python

- Learning Steps:

1. Rust syntax basics and differences from Python
2. Understanding static typing vs. dynamic typing
3. Rust variable declarations and primitive types
4. Control structures and how they differ from Python
5. Verification: Write a simple program converting a Python script to Rust
6. Use my newly acquired knowledge to build a simple yet impressive Rust CLI Tool for My

AI Output:

Great start You've framed a clear progression from Python into Rust, and it's smart that you want to ground it in a practical CLI project. Let's refine and expand your Phase 1 plan so it's both structured and achievable:

Mastering Rust Coming from Python

Environment: Ubuntu 24.04 LTS

End Goal: Develop proficiency in Rust and build a simple yet impressive CLI tool.

Phase 1: Language Fundamentals

Prerequisite: Comfortable programming knowledge in Python

Learning Steps

Rust syntax basics and differences from Python

Install Rust (rustup, cargo) and set up VS Code / preferred editor.

Learn Rust's main function, semicolons, and block structure.

Compare: Python's whitespace vs. Rust's braces {}.

Understanding static typing vs. dynamic typing

Python: types inferred at runtime.

Rust: types enforced at compile time, but with inference (let x = 5;).

Explore: let, let mut, constants, and immutability.

Rust variable declarations and primitive types

Integers (i32, u32), floats, booleans, chars, strings, and arrays.

Practice: Declare variables with and without type annotations.

Compare with Python's duck typing.

Control structures and how they differ from Python

if, else if, else (expressions in Rust).

match vs. Python's if/elif.

Loops: loop, while, for.

Example: FizzBuzz in Rust vs. Python.

Verification: Convert a Python script to Rust

Choose a small script (e.g., temperature converter, word counter, or simple calculator).

Translate line-by-line into Rust.

This highlights: explicit typing, ownership, error handling.

Capstone Step: Build a simple Rust CLI tool

Use the clap or argh crate for command-line argument parsing.

Ideas: To-do list manager, File renamer, Currency/Unit converter, Weather fetcher (using API)

Showcase clean code, error handling, and documentation.

Deliverable for Phase 1: A GitHub repo with Converted Python → Rust script, a working CLI tool with README.md.

Reflection:

The "Creating a Learning Journey Plan" helped me by structuring a study plan I could follow. After reading each topic, I answered short assignments to test my understanding, which made my learning more active and practical.

B) Step 2 — Project README Generation Prompt

Link:

https://ai.moringaschool.com/ai-software/ai-use-cases/usecases-documentation/#_3_user_guides_and_readme_files

Prompt Used:

Please create a comprehensive README.md file for my project based on the following information:

Project Title: Rust Calculator – A Fun, Stylish Rust CLI Tool

Objective: Document a guided, prompt-powered learning journey into Rust (from Python) on Ubuntu 24.04 LTS, culminating in a working CLI calculator that greets the user, remembers their name, and supports basic arithmetic.

Sections to include in README.md:

1. Project Title and Objective
2. Installation Instructions
3. System Requirements
4. Usage Instructions (with examples)
5. Minimal Working Example Code
6. Common Issues and Fixes
7. References / Links
8. Codebase Structure
9. License Information

Additional Notes:

- Keep formatting clean and professional.
- Include markdown headings, code blocks, and lists where appropriate.
- Ensure that all prompts and AI outputs are documented clearly.
- Include step-by-step instructions for running the Rust CLI tool.

End Goal: A complete README.md that can guide a beginner from setup to running the Rust Calculator, including explanations, example inputs/outputs, and troubleshooting tips.

AI Output:

```
# Rust Calculator
```

A **fun, stylish Rust calculator with personality**! This project goes beyond simple arithmetic by interacting with the user, remembering their name, and making math feel engaging with a touch of humor and emojis.

```
---
```

```
## Features
```

- Welcomes the user with a personalized greeting.
- Asks for the user's name and remembers it during calculations.
- Addition
- Subtraction
- ✖ Multiplication
- Division (with division-by-zero error handling).
- Fun, stylish CLI experience with emojis.

Technologies Used

- [Rust](https://www.rust-lang.org/)
- Standard Library: ``std::io``

Installation Requirements

1. **Install Rust**

[Rust Installation Guide](https://www.rust-lang.org/tools/install)

2. **Install VS Code Rust Extension**

[Rust Analyzer for VS Code](https://code.visualstudio.com/docs/languages/rust)

3. **Set up Cargo project**

```
```bash
cd path/to/folder
cargo new rust_calculator
cd rust_calculator
cargo build
cargo run
```
```

Usage Example

```
```bash
$ cargo run
Hello from Davids First Rust Calculator!
I was created on September 3rd 2025
```

Whats Your Name σ?

> David

Hello David ! Lets do some Maths !

Please enter the first number (e.g., 1, 2, 3, 4)

> 10

Please Enter an operator (e.g., , , \*, /):

> \*

Please enter the second number (e.g., 5, 6, 7, 8)

> 5

David The Result is : 50

```
```
```

☼ Configuration Options

Currently, the calculator runs interactively in the terminal. Future enhancements could include:

- Loop mode (do multiple calculations until "exit").
- CLI arguments (calculate directly via `cargo run -- 5 + 7`).

Known Issues

Emojis with non-yellow skin tones may display oddly in some terminals.

Use the default yellow skin tone for best compatibility.

Resources Used

- The Rust Book
- Cargo Book
- Tech With Tim Rust YouTube Playlist
- Get Emoji

Code Structure

```
rust_calculator/  
├── src/  
│   └── main.rs          # Main program source  
├── target/              # Auto-generated build artifacts  
│   ├── debug/  
│   ├── .rustc_info.json  
│   └── CACHEDIR.TAG  
├── .gitignore  
├── Cargo.lock  
├── Cargo.toml  
└── README.md  
```\n
```

## ## Troubleshooting

- Error: command not found: cargo → Ensure Rust and Cargo are properly installed and added to your PATH.
- Program crashes when entering text instead of a number → Input parsing requires numeric values (e.g., 10, 3.5).
- Emojis look strange in CLI → Some terminals may not support all emojis. Stick to basic ones.

## ## Contributing

Contributions are welcome! To contribute:

- Fork the repository.
- Create a new branch (feature/amazing-feature).
- Commit your changes.
- Push to the branch.
- Open a Pull Request.

## ## License

This project is open source under the MIT License. You are free to use, modify, and distribute it as you wish.

## Reflection:

This output saved time and ensured my documentation covered features, setup, usage, troubleshooting, and open-source etiquette.

## C) Example Rust Code

```
use std::io;

fn main() {
 println!("Enter your name:");
 let mut name = String::new();
}
```

```

io::stdin().read_line(&mut name).expect("Failed to read line");
println!("Hello, {}!", name.trim());

println!("Enter first number:");
let mut first = String::new();
io::stdin().read_line(&mut first).expect("Failed to read line");
let first: i32 = first.trim().parse().expect("Please type a number!");

println!("Enter second number:");
let mut second = String::new();
io::stdin().read_line(&mut second).expect("Failed to read line");
let second: i32 = second.trim().parse().expect("Please type a number!");

println!("{}", first + second, first, second, first + second);
}

```

## 7) Common Issues & Fixes

- Error: cargo: command not found → Ensure Rust and Cargo are installed and added to PATH.
- Error: Invalid input like 'ten' → Only numeric input allowed.
- Issue: Some terminals don't display special characters properly.

## 8) References

- Rust Book → <https://doc.rust-lang.org/book/title-page.html>
- Cargo Book → <https://doc.rust-lang.org/stable/cargo/index.html>
- TechWithTim → [https://youtube.com/playlist?list=PLzMbGfZo4-nyLTISRbVo0zjSnCnqjHYQ&si=JNHu\\_og204tQpfyc](https://youtube.com/playlist?list=PLzMbGfZo4-nyLTISRbVo0zjSnCnqjHYQ&si=JNHu_og204tQpfyc)

## 9) Codebase Structure

```

rust_calculator/
├── src/
│ └── main.rs
├── target/
│ ├── debug/
│ └── .rustc_info.json
├── CACHEDIR.TAG
├── .gitignore
├── Cargo.lock
├── Cargo.toml
└── README.md

```