Your first day with the Rust programming language

By Said Aroua
CH Open Workshop-Days 2025

Agenda

- Introduction
  - `whoami`
  - *who R U*
- A Tour of Rust
  - Why Rust
  - Tooling
  - Syntax and common types
  - Ownership
- Installing Rust
- Rustlings
- *Lunch*
- *Lots of experimenting and learning*
- Sharing experiences
- Closing words

`whoami`

- Said, born '97
- Martial arts, Jiu Jitsu right now
- Scouts
- Rock climbing
- Gaming

About me, professionally

- Highschool
- One semester of Computer Science @ ETH
- Vocational training as application developer (2y)
- + two more years of mostly .NET development
- Three years @ HSLU BSc Computer Science

- Research assistant @ Applied AI lab, HSLU

Languages I got paid for

- C#
- Rust
- Python
- Typescript/JavaScript (Angular, React)
- SQL
- Java
- Go
- Some ungodly, handrolled scripting language in a 20 year old clinical information system

Languages I also tried

- Kotlin
- Lua
- Common LISP
- Clojure
- Prolog
- C
- C++
- x86 ASM
- Typst

My Rust journey so far (~500h)

- Interested since ~2019
  - Major influences: fasterthanlime, ThePrimeagen
- Student project with Applied Cybersecurity Lab @ HSLU
  - Implementing quantum-safe hashing algorithms for privacy amplification in Rust
- *Mobile Lab* seminar work: Rust on Android
  - Tiny cellular automaton in Rust, visualisation in Kotlin with Jetpack Compose
- Bachelor's thesis: *Integrating Post-Quantum Cryptography with a Modern Memory-Safe TLS Library: A Case Study of liboqs and Rustls*
- Small Open Source contributions
  - Typst compiler
  - `cargo-semver-checks`

Who R U

- Interest in Rust / motivation for the workshop
- Main technologies you know
- Current professional role

A tour of Rust

Origins

- Started in Mozilla Research 2009
- v1.0 in May 2015
  - Rust is officially 10 years old already!
- Governed through the Rust Foundation since 2021

Rust’s Strengths

- Performance similar to C
- Memory safety, no garbage collector
- Rich type system
- Growing ecosystem
- Modern tooling: **Cargo**
  - Helpful compiler
  - Formatter
  - Clippy (linter)
  - Package management and build system
  - Cross compilation

> It’s time to halt starting any new projects in C/C++ and use Rust for those scenarios where a non-GC language is required. For the sake of security and reliability, the industry should declare those languages as deprecated.

Mark Russinovich, Microsoft Azure CTO, 2022

70% of all critical security vulnerabilities at Microsoft and in Chromium are due to **memory safety problems**.

Rust doesn’t have these problems.

Where Rust shines

- Command line tools
  - `ripgrep`: Rust `grep` variant with ~8x better performance
- WebAssembly
  - Run in the browser or on a serverless platform
- High performance systems
  - Rust powers big parts of Figma since 2018
- Embedded or in the Linux kernel

Other interesting corners of Rust development

- Game engines, e.g. Bevy
- GPU programming, e.g. RustGPU
- Typst document markup language and compiler (LaTex-like)
- Lots of interesting databases
  - SurrealDB multi-model database
  - Turso SQLite reimagined
  - SpaceTimeDB Code and DB in one
- ... and many more

Rust Terms/Concepts

- Crates -> packages
- Cargo -> build tool & package manager
- Rust Editions -> "major Rust versions", very few breaking changes
- Rust nightly -> Nightly builds of Rust with bleeding edge features
  - Some people use nightly for serious projects, though this isn’t recommendable
- Traits -> "interfaces"
- Ownership, Borrowing, and Lifetimes -> Concepts of Rust’s unique model that ensures memory safety

Variables

```
// Declaration with assignment, once with inferred type
let some_number = 42;
let another_number: i32 = 42;

// This line won't compile, immutable by default
another_number = 21;

// Marking variables mutable EXPLICITLY with "mut"
let mut counter = 0;
counter += 1;
```

Branching

```
// Notice no parens for condition (like Python)
if true {
  foo();
} else {
  bar();
}

// if is also an expression (better ternaries)
let conditional = if condition { true_value } else { false_value };

// Similar to "switch" in C-like languages but much more powerful
// Pattern matching like functional languages
// Also an expression like "if", exhaustiveness checks by the compiler
match 0 {
  0 ⇒ foo(),
  _ ⇒ bar() // catch all default case with underscore
}
```

Looping

```
// Ranges
for i in 0..5 { // could also write 0..=4 here
}

// Iteration on arrays (and many other collections and iterators)
for i in [1, 2, 3] {
}

// "loop" just loops forever
loop {
  foo();
  if exit_condition {
    break;
  }
}
```

Functions

```
fn greet() {
  println!("hello world"); // ! signifies that println is a macro
}

pub fn add(left: i32, right: i32) -> i32 {
  left + right // notice no semicolon! -> implicit return on last expression
}
```

Enums

```
pub enum Option<T> { // in stdlib, Maybe type in Haskell
  None,
  Some(T), // variant specific data
}

// common pattern: matching on enum values
// compiler forces exhaustive checks
match Some(42) {
  Some(value) ⇒ value,
  None ⇒ unreachable!(),
}
```

Structs

```
#[derive(Clone)]
struct CPU {
  name: String,
  max_clock_hz: u64,
  core_count: u32,
}

impl CPU {
  /// Triple slashes are doc comments
  fn new(name: String, max_clock_hz: u64, core_count: u32) -> Self {
    Self { name, max_clock_hz, core_count }
  }

  fn rename(&mut self, new_name: String) {
    self.name = new_name;
  }
}

// Consuming code
let mut cpu = CPU::new(name, hz, cores);
cpu.rename("skylake")
```

Traits

```
trait Codec {
  type Encoding; // advanced: associated types

  fn encode(data: &[u8]) -> Self::Encoding;
  fn decode(encoded: Self::Encoding) -> Vec<u8>;
}

impl Codec for CPU {
  // TODO: Define the associated type and implementation
}

// Compile-time known specialisation
fn wrap_decode(decoder: impl Codec) {}
// Dynamic, runtime decided codec
fn wrap_decode_dyn(decoder: Box<dyn Codec<Encoding = SomeEncoding>>) {}
```

Essential containers

```
Option<T> // Some(T), None
Result<T,E> // Ok(T), Err(E)
Vec<T>, vec![] // dynamic size array
[T; 8] // fixed size array
&[T] // readonly view of collection
Iterator<T> // hopefully needs now introduction
```

Ownership in Rust

Problem in C: Use after free

```c
char* data = (char*)malloc(64);
strcpy(data, "This is some data stored in heap memory!");
printf("Data: %s\n", data);
free(data);
// Use After Free: undefined behavior
printf("Data after free: %s\n", data);
```

Here the error is obvious.

Applied to the Linux kernel which has recently surpassed 40 Million lines of code it's not.

Rust’s Ownership Model

```
let message = "Hello World".to_owned();
print!("{}", message);
drop(message); // think of "drop" as Rust's "free"
print!("{}", message); // Compiler disallows this line
// Compiler error: borrow of moved value: `message`
```

Rust’s Ownership Model

```
let message = "Hello World".to_owned();
print!("{}", message);
// now it compiles
print!("{}", message);

// Compiler automatically inserts this
// after message falls out of scope (e.g. at the end of a function)
drop(message); // we rarely use `drop` directly
```

Borrowing

```
let some_big_array = [0u8; 4096];
borrowing(&some_big_array);
owning(some_big_array);
```

Lifetimes

https://doc.rust-lang.org/rust-by-example/scope/lifetime.html

Any questions so far?

Installing Rust

Any editor of choice
Simple recommendation: VSCode + rust-analyzer extension

Rustlings

Small, focused exercises to get to know Rust

Keep in mind

- Learning from scratch takes some patience
- The compiler is strict but very helpful
- LLM output as guidance, not gospel
- Ask me anything
- Maybe ask your neighbor if I'm currently occupied
- Essential Rust learning resources:
  - Official Rust Book
  - Rust by Example

Time to start a project

Ideas:

- Advent of Code
- Sudoku solver
- Terminal color printer
- Any of your own ideas
  - Avoid networking and async for the beginning

Reflections

- What was great about getting to know Rust?
- Where did you get stuck or frustrated?
- Do you think you have enough of an idea of Rust to maybe use it?

Connect with me

- Email: said.aroua@bluewin.ch
- LinkedIn: https://www.linkedin.com/in/said-aroua/

Available for work ~ June 2026

Merci!