A brief tour of Rust

c@pgdm.ch

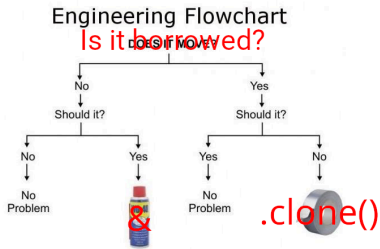
January 16, 2025

Contents

Contents

Goals of the presentation

- Give an overview of the Rust programming language and discuss how we have been using it at $J0B.
- "Live" coding of a sequence alignment tool!
- Discussion / Q&A



Side goal: understand this joke (credit Tim and dullhunk)

Contents

Caveat lector

- I'm no programming language extremist. I also work with Python and C++ to great success.
  - I would never train NNs or write games in Rust (again).
  - I was originally pretty annoyed with the whole Rust hype.
- These slides are written in Rust (I know), with Typst.

Contents

Using Rust

- We have been using Rust at `$JOB` since July 2021.
  - Average of 10-15 concurrent contributors, from interns to experienced SWE and MLE.
- Technical stack:
  - Python for defining and training NNs (PyTorch).
  - C++ for the real-time software (including NN inference).
  - Rust for the rest: data processing, ML lifecycle management, evaluation, pipelines, services...
- Why Rust?
  Development velocity of Python, with performance and development-time guarantees of C++ (and beyond).

Contents

Rust overview

https://www.rust-lang.org/

- First version in 2012, now at 1.84 (release every 6 weeks).
- Backed by the Rust Foundation. MIT & Apache 2.0 license.
- Emphasis on performance, concurrency, type safety, memory safety, and developer experience.
- Wide and expanding adoption:
  - Industry: Amazon, Google, Meta, Microsoft, Cloudflare, Github, Apple, Huawei, Discord...
  - Open source: Servo, uv, Ripgrep, Wezterm, Typst, Zed, Helix, ruff, Sccache, Hyperfine, Alacritty, Polars, InfluxDB, Meilisearch, Deno, Linux kernel...

Rust found a sweet spot: it is just as low-level as C or C++ with all the advantages of these (e.g. control, size, speed, etc.) At the same time, it is as high-level as Haskell with an amazing amount of functional heritage. It is still imperative, so quite accessible to most people, and it is just as flexible as Python.

— *Peter Varo*

Rust overview

- Basic syntax similar to C++.
- Incorporates best ideas from other languages:
  - Algebraic data types
  - Immutability by default
  - Pattern matching
  - Move semantics
  - Expression-orientation
  - Traits-based OOP
  - Functional programming
  - Zero-cost abstractions
  - Ergonomic error handling
  - Asynchronous programming
  - Generics
  - Type inference

  (examples later)
- Many supported targets, including WebAssembly. LLVM and gcc backends.

Rust overview

- Excellent built-in tooling:
  - `cargo` (build system and dependency management)
  - `clippy` (linter)
  - `fmt` (formatter)
  - `rust-analyzer` (LSP)
  - `rustdoc` (HTML documentation)
  - `rustc` (compiler), with *excellent* errors

  Nice consequence: uniformity and compatibility throughout the ecosystem.
- Rich and well-documented standard library (example).
- Interoperability with other languages (e.g. C, Python via `pyo3`, C++ via `cxx`).

Rust overview

- High-quality libraries, centralized on crates.io (similar to pip). For example:
  - `serde` ((de)serialization)
  - `rayon` (parallel iterators/thread pool)
  - `regex` (regex engine)
  - `clap` (command line arguments)
  - `tracing` (structured logging)
  - `tower` (networking)
  - `axum` (web framework)
  - `tokio` (async runtime)
- Plenty of excellent learning resources:
  - The Rust Book
  - Rust by Example
  - *Programming Rust* book
  - https://users.rust-lang.org/

Rust overview

First binary:

```rust
fn main() {
    let who = "Ferris";
    // Or without using type inference:
    // let who: &str = "Ferris";
    println!("Hello {}!", who);
}
```

```
$ cargo run -r
Hello Ferris!
```

Rust overview

Variables are immutable by default:

```rust
fn main() {
    let who = "Ferris";
    who = "Not Ferris";
    println!("Hello {}!", who);
}
```

```
error[E0384]: cannot assign twice to immutable variable `who`
 --> immutability.rs:3:5
  |
2 |     let who = "Ferris";
  |         --- first assignment to `who`
3 |     who = "Not Ferris";
  |     ^^^^^^^^^^^^^^^^^^ cannot assign twice to immutable variable
  |
help: consider making this binding mutable
  |
2 |     let mut who = "Ferris";
  |         +++
```

(note the nice compiler errors!)

Rust overview

A *mutable* reference on a variable must be the unique reference on it.

The *borrow checker* enforces this, one of Rust's keys to guaranteeing memory safety.

```rust
fn main() {
    let mut v: Vec<i64> = vec![1, 2, 3];
    for vv in &v {
        v.push(4); // Ouch, we modify v while iterating on it!
    }
}
```

```
error[E0502]: cannot borrow `v` as mutable because it is also borrowed as immutable
 --> borrow_checker.rs:5:13
  |
3 |     for vv in &v {
  |               --
  |               |
  |               immutable borrow occurs here
  |               immutable borrow later used here
4 |         if vv % 2 == 0 {
5 |             v.push(4);
  |             ^^^^^^^^^ mutable borrow occurs here
```

Rust overview

The borrow checker also checks for reference lifetimes:

```rust
// This returns a reference to a temporary object...
fn dodgy() -> &Vec<u8> {
    &vec![1, 2, 3]
}
fn main() {
    let v = dodgy();
}
```

```
error[E0515]: cannot return reference to temporary value
 --> borrow_checker2.rs:2:5
  |
2 |     &vec![1, 2, 3]
  |     ^-------------
  |     ||
  |     |temporary value created here
  |     returns a reference to data owned by the current function
```

Rust overview

On the other hand, this is fine:

```rust
fn valid(input: &Vec<u8>) -> &Vec<u8> {
    &input
}
fn valid2(input: &Vec<String>) -> Option<&String> {
    input.first()
}

fn main() {
    let v = vec![1, 2, 3];
    let v2 = valid(&v);

    let v = vec![String::from("test")];
    let v2 = valid2(&v);
}
```

(the compiler is automatically inferring lifetimes for us)

Rust overview

Move semantics are the default.

```rust
fn f(v: Vec<i64>) {}
fn main() {
    let v = vec![1, 2, 3];
    f(v); // v is moved into f
    println!("{}", v[0]); // invalid, v was moved!
}
```

```
error[E0382]: borrow of moved value: `v`
 --> move_semantics.rs:5:20
  |
3 |     let v = vec![1, 2, 3];
  |         - move occurs because `v` has type `Vec<i64>`, which does not implement the `Copy` trait
4 |     f(v);
  |       - value moved here
5 |     println!("{}", v[0]);
  |                    ^ value borrowed here after move
  |
note: consider changing this parameter type in function `f` to borrow instead if owning the value isn't necessary
 --> move_semantics.rs:1:9
  |
1 | fn f(v: Vec<i64>) {}
  |    -    ^^^^^^^^ this parameter takes ownership of the value
  |    |
  |    in this function
help: consider cloning the value if the performance cost is acceptable
  |
4 |     f(v.clone());
  |        ++++++++
```

Rust overview

```rust
fn f(v: Vec<i64>) {} // Move
fn f_ref(v: &Vec<i64>) {} // Reference
fn f_mut_ref(v: &mut Vec<i64>) {} // Mutable reference

fn main() {
    let v = vec![1, 2, 3];

    f(v.clone()); // Clone v explicitly
    f_ref(&v); // Pass v by reference

    let mut v = v; // Rebind v to be mutable
    f_mut_ref(&mut v); // Pass v by reference

    for vv in &v {} // References
    for vv in &mut v {} // Mutable references
    for vv in v {} // v is moved
}
```

The copy of expensive types must be done explicitly, via the `clone` method.

No implicit copy like in C++.

Rust overview

Powerful pattern matching:

```rust
fn print_age(age: Option<u8>) {
    match age {
        Some(age) if age < 150 => {
            println!("{}", age)
        }
        None => println!("No age provided"),
        _ => println!("Invalid age"),
    }
}
fn main() {
    print_age(None);
    print_age(Some(10));
    print_age(Some(255));
}
```

`Option<T>` is a generic datatype, corresponding to an enum taking values `None` or `Some(T)`.

Rust overview

Enums and pattern matching:

```
enum Entity {
    Person {
        first_name: String,
        last_name: String,
        age: u8,
    },
    Company {
        name: String,
    },
    Custom(String), // Tuple variant
    Unknown,        // Unit variant
}
fn f(e: &Entity) {
    match e {
        Entity::Company { name } => {}
        _ => {}
    }
}
```

Rust overview

Error handling with `Result<T,E>`:

```rust
// A function that returns a result
fn double(x: i32) -> Result<i32, ()> {
    if x == 10 {
        return Err(());
    }
    Ok(2 * x)
}
// An error will translate in a non-zero exit code
fn main() -> Result<(), ()> {
    // Note that ? operator
    let r: i32 = double(1)? + double(2)?;
    Ok(())
}
```

```rust
let x = r?; // is syntactic sugar for:
let x = match r { Ok(r) => r, Err(e) => { return Err(e)} };
```

Rust overview

Iterators and functional programming:

```rust
fn main() {
    let s = (0..100_i64)
        .filter(|i| i % 2 == 0)
        .map(|i| [i.pow(2), i.pow(3)])
        .flatten()
        .sum::<i64>();

    let x: Option<i64> = Some(10);
    let x: i64 = x.map(|x| x * 2).unwrap_or(2);
}
```

Very convenient for manipulating `Iterators` (map, filter, flatten, etc.), `Options`, `Results`...

Async (streams) and parallel (`rayon`) variants as well.

Rust overview

Defining a struct and methods:

```rust
#[derive(Debug)] // Derive macro
struct Person {
    name: String,
    age: u8,
}
impl Person {
    fn can_vote(&self) -> bool {
        self.age >= 18
    }
}
fn main() {
    let p = Person {
        name: "Linus".into(),
        age: 55,
    };
    // Prints Person { name: "Linus", age: 55 } true
    println!("{:?} {:?}", p, p.can_vote());
}
```

Rust overview

Traits and generics:

```rust
struct Person {
    name: String,
    age: u8,
}
trait Entity {
    fn identifier(&self) -> &String;
}
impl Entity for Person {
    fn identifier(&self) -> &String {
        &self.name
    }
}
fn call(e: &impl Entity) {
    println!("{}", e.identifier());
}
// Alternatively:
fn call2<E: Entity>(e: &E) {}
```

Rust overview

An incredibly useful crate: `serde`.

```rust
use serde::{Deserialize, Serialize};

#[derive(Debug, Serialize, Deserialize)]
struct Person {
    name: String,
    age: u8,
    skills: Vec<String>,
}
#[derive(Debug, Serialize, Deserialize)]
struct Persons(Vec<Person>);

fn save(p: &Persons) -> serde_json::Result<()> {
    std::fs::write("out.json", &serde_json::to_string_pretty(&p)?);
    Ok(())
}
```

This enables serialization and deserialization in JSON, CSV, YAML, CBOR, Bincode, TOML, Pickle, etc.

New formats can be easily implemented (by implementing a trait), as well as custom handling of types.

Rust overview

Clippy performs powerful static analysis, which helps the user write idiomatic, performant, and bug-free code.

```rust
struct Foo(f32);
impl std::ops::Add for Foo {
    type Output = Foo;
    fn add(self, other: Foo) -> Foo {
        Foo(self.0 - other.0)
    }
}
fn main() {}
```

```
warning: suspicious use of `-` in `Add` impl
 ⟶ clippy.rs:6:20
  |
6 |         Foo(self.0 - other.0)
  |                    ^
  |
  = help: for further information visit https://rust-lang.github.io/rus
  = note: `#[warn(clippy::suspicious_arithmetic_impl)]` on by default
```

See all lints on https://rust-lang.github.io/rust-clippy/master/

Rust overview

Leftover

- Documentation (`rustdoc`).
- Concurrent execution: Rust's memory management allows *fearless concurrency*.
  The language does not prevent deadlocks, but it is impossible to create memory safety issues (e.g. access from two threads) in safe Rust. Remember that it is not possible to take a mutable reference while there exist immutable references, and vice-versa.
- Asynchronous programming: https://c.pgdm.ch/notes/practical-async-rust-talk/
- Check out the crates available on https://crates.io See also https://blessed.rs/crates for a curated list.
- Zero-dependency binaries (compile against an old `glibc` and use `rustls` instead of `openssl`).
- Cross-compilation.

Rust overview

Negative aspects

- `cargo check` is fairly fast to run, even on very large codebases, but building in release mode can take a while. Multiple optimizations are possible (disable LTO, enable incremental, caching in CI...).
- The learning curve is steeper than with Python, especially when working with concurrent code.
  In our experience, it takes people roughly 2-3 weeks to be fully productive (working in an existing codebase is easier than starting from scratch, though).
- The borrow checker requires different thinking and adapted architectures. For beginners, this means that prototyping ideas will be slow and possibly painful.
- While the language is already fairly mature, it is rapidly evolving and some features are still under development (e.g. native async traits).
- Dependency creep, due to the easy of adding them and transitivity.
- Often adds to tech stacks with multiple other languages (Python, C++, Go...).

Contents

“Live” coding

Implement a small binary performing sequence alignment of FASTA files.

- Command line parsing with `clap`.
- Parallel processing with `rayon`.
- Serialization with `serde`.
- Structured logging with `tracing`.
- HTTP mode with `axum`.
- (Python integration)

You can also follow along on https://github.com/cpg314/sequence-alignment for the full source code.

To install Rust: https://www.rust-lang.org/learn/get-started (install `rustup` and pull the latest toolchain).

“Live” coding

```
$ alignment -h
Usage: alignment [OPTIONS] <COMMAND>

Commands:
  align  Align the first two sequences in a FASTA file
  serve  Launch alignment HTTP service
  help   Print this message or the help of the given subcommand(s)

Options:
      --mismatch-penalty <MISMATCH_PENALTY>  [default: -2]
      --gap-penalty <GAP_PENALTY>            [default: -1]
  -h, --help                                 Print help
```

Align HBB_HUMAN with HBB_HORSE (from UniProt):

```
2025-01-15T22:53:54.505972Z  INFO from_path{p="sequences.fasta"}: alignment::fasta: Parsing FASTA file
2025-01-15T22:53:54.506568Z  INFO alignment: Aligning
"sp|P68871|HBB_HUMAN Hemoglobin subunit beta OS=Homo sapiens OX=9606 GN=HBB PE=1 SV=2" with
"sp|P02062|HBB_HORSE Hemoglobin subunit beta OS=Equus caballus OX=9796 GN=HBB PE=1 SV=1"
2025-01-15T22:53:54.506714Z  INFO alignment: Alignment with score -49.00, 71.76% aligned
VH-LTP--EEKSA-VT-ALWG-KVNVDE--VGGEALGRLLVVYPWTQRFFE-SFGDLST-PD-AVMGNPKVKAHGKKVLGAF-SD-GLA---H-LDNLKGTFAT-LSELHCDKLHVDPENFRLLGNVLVCV-LA-HHFGKE-FTPPV--QAA-YQKVVAGVANALAHKYH
V-QL--SGEEK-AAV-LALW-DKVN--EEEVGGEALGRLLVVYPWTQRFF-DSFGDLS-NP-GAVMGNPKVKAHGKKVL---HS-FG--EGVHHLDNLKGTFA-ALSELHCDKLHVDPENFRLLGNVLV-VVLARH-FGK-DFTP--ELQA-SYQKVVAGVANALAHKYH
2025-01-15T22:53:54.506723Z  INFO alignment: Performed 1 run(s) in 735.355µs (1360 runs/s)
```

“Live” coding

Representing sequences:

```rust
/// A sequence represented as a list of `T`
#[derive(Deserialize, Debug, Default)]
pub struct Sequence<T = char>(pub Vec<T>);
```

Representing a FASTA file:

```rust
/// A single sequence with metadata
#[derive(Debug)]
pub struct FastaSequence<T> {
    pub meta: String,
    pub sequence: Sequence<T>,
}

/// A decoded FASTA file as a list of sequences
#[derive(Debug)]
pub struct Fasta<T>(pub Vec<FastaSequence<T>>);
```

“Live” coding

```rust
impl<T: From<char>> Fasta<T> {
    #[tracing::instrument]
    pub fn from_path(p: &Path) -> anyhow::Result<Self> {
        info!("Parsing FASTA file");
        let data = std::fs::read_to_string(p)?;
        let lines = data.lines();
        let mut sequences = vec![];
        for line in lines {
            if let Some(meta) = line.strip_prefix(">") {
                sequences.push(FastaSequence {
                    meta: meta.into(),
                    sequence: Default::default(),
                });
            } else {
                match sequences.last_mut() {
                    Some(l) => l.sequence.0.extend(line.chars().map(T::from)),
                    None => anyhow::bail!("Sequence without metadata"),
                }
            }
        }
        Ok(Self(sequences))
    }
}
```

“Live” coding

Representing alignments :

```rust
/// An alignment of two sequences
#[derive(Debug, Serialize, Deserialize)]
pub struct Alignment<T> {
    pub alignment: VecDeque<[Option<T>; 2]>,
    /// Note that the score depends on the aligner parameters
    score: f32,
}

impl<T: PartialEq> Alignment<T> {
    /// Ratio of the number of aligned pairs divided by the length including gaps
    fn matching_ratio(&self) -> f32 {
        self.alignment
            .iter()
            .filter(|[a, b]| a.is_some() && a == b)
            .count() as f32
            / self.alignment.len() as f32
    }
}
```

“Live” coding

```rust
#[derive(Parser, Debug, Copy, Clone)]
pub struct Aligner {
    #[clap(long, default_value_t=-2.0)]
    mismatch_penalty: f32,
    #[clap(long, default_value_t=-1.0)]
    gap_penalty: f32,
}

impl Aligner {
    /// Align two sequences and return an alignment

    pub fn align<T: std::cmp::PartialEq + Copy>(&self, seqs: [&Sequence<T>; 2]) -> Alignment<T> {
      // Needleman-Wunsch
    }
```

```rust
let fasta = fasta::Fasta::<char>::from_path(fasta)?;
anyhow::ensure!(fasta.0.len() == 2, "Expecting exactly two sequences");
let seq1 = &fasta.0[0];
let seq2 = &fasta.0[1];
info!("Aligning {:?} with {:?}", seq1.meta, seq2.meta);
let alignment = args.aligner.align([&seq1.sequence, &seq2.sequence]);
info!("{}", alignment);
```

“Live” coding

Serialize alignments with Serde:

```rust
#[derive(Debug, Serialize, Deserialize)]
pub struct Alignment<T> {
    pub alignment: VecDeque<[Option<T>; 2]>,
    /// Note that the score depends on the aligner parameters
    score: f32,
}
impl<T: Serialize> Alignment<T> {
    pub fn write(&self, filename: &Path) -> anyhow::Result<()> {
        // Swap JSON for your favourite format (e.g. bincode, cbor...)
        Ok(std::fs::write(filename, serde_json::to_string(&self)?)?)
    }
}
```


```
{"alignment":[["V","V"],["H",null],[null,"Q"],["L","L"],["T",null],["P",null],[null,"S"],[null,"G"],["E","E"],["E","E"],["K","K"],["S",null],["A","A"],[null,"A"],["V","V"],["T",null],[null,"L"],["A","A"],["L","L"],["W","W"],["G",null],[null,"D"],["K","K"],["V","V"],["N","N"],["V",null],["D",null],["E","E"],[null,"E"],[null,"E"],["V","V"],["G","G"],["G","G"],["E","E"],["A","A"],["L","L"],["G","G"],["R","R"],["L","L"],["L","L"],["V","V"],["V","V"],["Y","Y"],["P","P"],["W","W"],["T","T"],["Q","Q"],["R","R"],["F","F"],["E","E"],["E",null],[null,"D"],["S","S"],["F","F"],["G","G"],["D","D"],["L","L"],["S","S"],["T",null],[null,"N"],["P","P"],["D",null],[null,"G"],["A","A"],["V","V"],["M","M"],["G","G"],["N","N"],["P","P"],["K","K"],["V","V"],["K","K"],["A","A"],["H","H"],["G","G"],["K","K"],["K","K"],["V","V"],["L","L"],["G",null],["A",null],["F",null],[null,"H"],["S","S"],["D",null],[null,"F"],["G","G"],["L",null],["A",null],[null,"E"],[null,"G"],[null,"V"],["H","H"],[null,"H"],["L","L"],["D","D"],["N","N"],["L","L"],["K","K"],["G","G"],["T","T"],["F","F"],["A","A"],["T",null],[null,"A"],["L","L"],["S","S"],["E","E"],["L","L"],["H","H"],["C","C"],["D","D"],["K","K"],["L","L"],["H","H"],["V","V"],["D","D"],["P","P"],["E","E"],["N","N"],["F","F"],["R","R"],["L","L"],["L","L"],["G","G"],["N","N"],["V","V"],["L","L"],["V","V"],["C",null],["V","V"],[null,"V"],["L","L"],["A","A"],[null,"R"],["H","H"],["H",null],["F","F"],["G","G"],["K","K"],["E",null],[null,"D"],["F","F"],["T","T"],["P","P"],["P",null],["V",null],[null,"E"],[null,"L"],["Q","Q"],["A","A"],["A",null],[null,"S"],["Y","Y"],["Q","Q"],["K","K"],["V","V"],["V","V"],["A","A"],["G","G"],["V","V"],["A","A"],["N","N"],["A","A"],["L","L"],["A","A"],["H","H"],["K","K"],["Y","Y"],["H","H"]],"score":-49.0}
```

“Live” coding

Command line arguments with clap:

```rust
#[derive(Parser)]
struct Flags {
    #[clap(flatten)]
    aligner: align::Aligner,
    #[clap(subcommand)]
    mode: Mode,
}
#[derive(Subcommand)]
enum Mode {
    Align {
        fasta: PathBuf,
        #[clap(long, short, default_value_t = 1)]
        runs: u32,
        #[clap(long, conflicts_with = "runs")]
        output: Option<PathBuf>,
    },
    Serve {
        #[clap(long, default_value_t = 3000)]
        port: u32,
    },
}
```

“Live” coding

Parallel processing with `rayon`:

```rust
(0..*runs)
    // Parallelism with rayon
    .into_par_iter()
    .try_for_each(|_| {
        let seq1 = &fasta.0[0];
        let seq2 = &fasta.0[1];
        info!("Aligning\n{:?} with\n{:?}", seq1.meta, seq2.meta);
        let alignment = args.aligner.align([&seq1.sequence, &seq2.sequence]);
        info!("{}", alignment);
        if let Some(output) = &output {
            alignment.write(output)?;
        }
        anyhow::Ok(())
    })?;
```

10^5 runs on the `HBB_HUMAN` / `HBB_HORSE` pair:

- Sequential: 8.5 seconds
- Rayon (thread pool): 2.5 seconds

“Live” coding

Web server with `axum`, with a JSON interface

```rust
#[derive(Deserialize)]
pub struct AlignData {
    seq1: String,
    seq2: String,
}
pub async fn align_post(
    axum::extract::State(aligner): axum::extract::State<align::Aligner>,
    axum::Json(data): axum::Json<AlignData>,
) -> axum::Json<align::Alignment> {
    let seq1 = Sequence::from(&data.seq1);
    let seq2 = Sequence::from(&data.seq2);
    axum::Json(aligner.align([&seq1, &seq2]))
}
```

```rust
let app = axum::Router::new()
    .route("/align", axum::routing::post(web::align_post))
    .with_state(args.aligner);
let listener = tokio::net::TcpListener::bind(format!("0.0.0.0:{}", port)).await?;
axum::serve(listener, app).await?;
```

“Live” coding

More things:

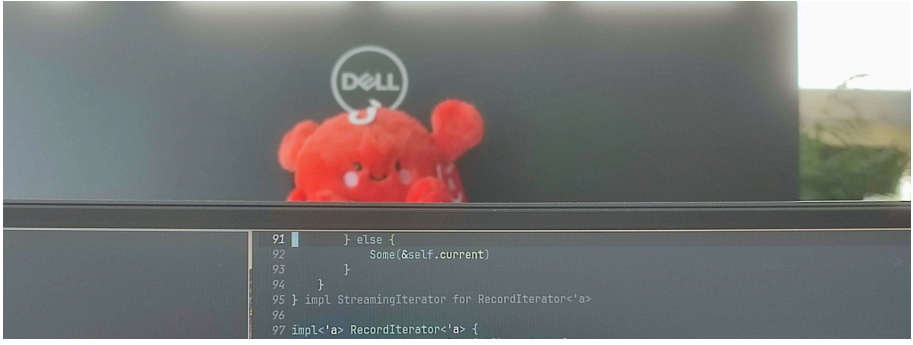
- Rustdoc

```
$ cargo doc --open
```

- Clippy

```
$ cargo clippy
```

Discussion


DELL
91 } else {
92 Some(&self.current)
93 }
94 }
95 } impl StreamingIterator for RecordIterator<'a>
96
97 impl<'a> RecordIterator<'a> {