LAAS CNRS

Rust

Bases et examples

2024-06-06

Guilhem Saurel

LAAS CNRS

This presentation

Available at

`https://homepages.laas.fr/gsaurel/talks/rust-intro.pdf`

Under License

This presentation (continued)

Source

`https://gitlab.laas.fr/gsaurel/talks : rust-intro.md`

Discussions & Companion project

https://github.com/nim65s/RobotS

Introduction

LAAS CNRS

Goals

1. introduction
2. nice features in a langage
3. nice features in the tooling
4. scope
5. limitations

Performance, Reliability, Productivity

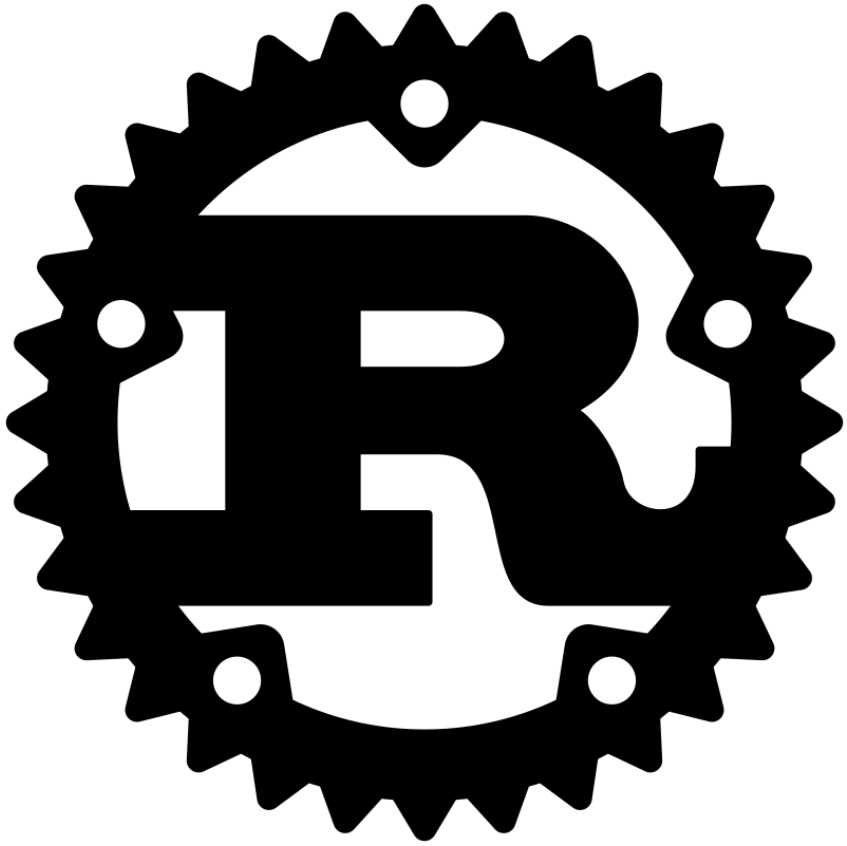

Figure 1: Rust

Base Tooling

Rustup

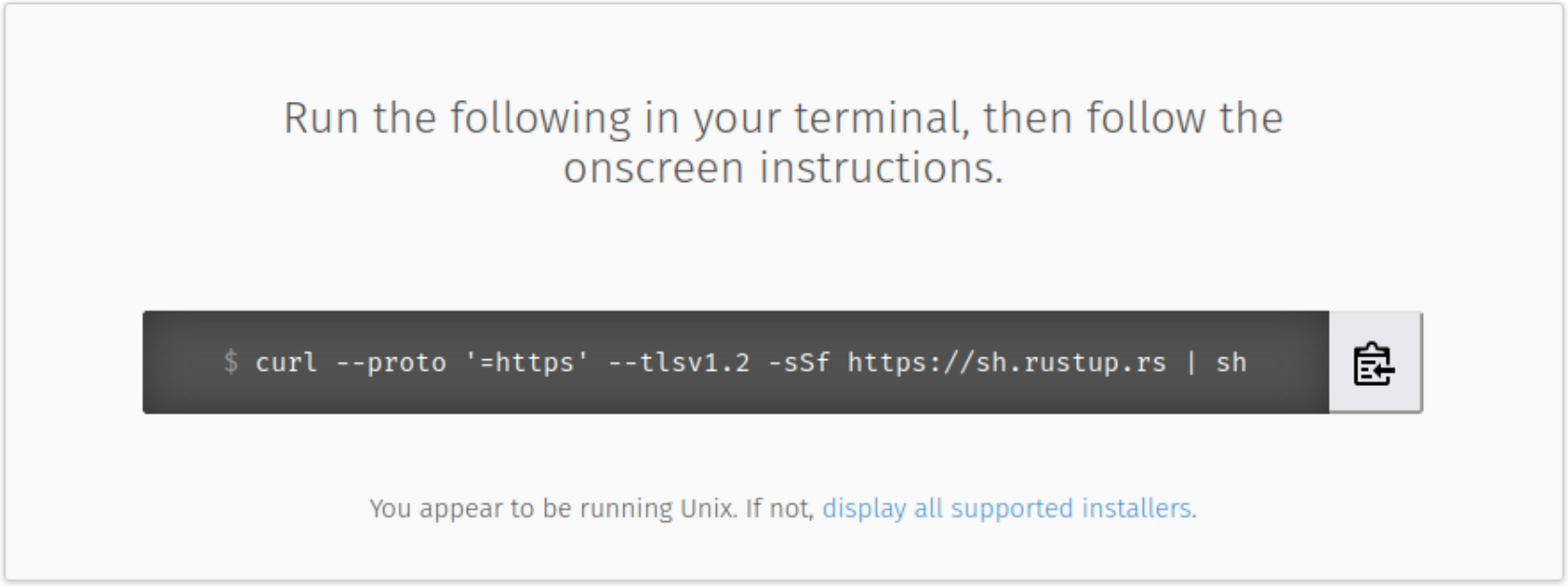

Figure 2: https://rustup.rs/

Cargo

```
$ mkdir first
$ cd first
$ cargo init
     Creating binary (application) package
$ cargo run
   Compiling first v0.1.0 (/root/first)
    Finished `dev` profile [unoptimized + debuginfo]
     Running `target/debug/first`
Hello, world!
```

Language

Hello World

```
$ cat src/main.rs

fn main() {
    println!("Hello, world!");
}
```

Mutability

```
let x = 5;
let mut y = 6;
```

Mutability

```
let x = 5;
let mut y = 6;

x += 1; // Error
y += 1; // Ok
```

Compiler Errors

```
Compiling first v0.1.0 (/tmp/first)
error[E0384]: cannot assign twice to immutable varia
  --> src/main.rs:27:5
   |
24 |     let x = 1;
   |         -
   |         |
   |         first assignment to `x`
   |         help: consider making this binding muta
...
27 |     x += 1;
   |     ^^^^^^ cannot assign twice to immutable var

For more information about this error, try `rustc --
error: could not compile `first` (bin "first") due t
```

Primitives: Scalar Types

- Signed integers: `i8`, `i16`, `i32`, `i64`, `i128` and `isize`
- Unsigned integers: `u8`, `u16`, `u32`, `u64`, `u128` and `usize`
- Floating point: `f32`, `f64`
- `char`: Unicode scalar values like `'a'`, `'α'` and `'∞'`
- `bool`: either `true` or `false`
- unit type: `()`

Primitives: Compound Types

- Arrays: `[1, 2, 3]`
- Tuples: `(1, true)`

Standard library

```
std::string::String
std::vec::Vec<T>
std::boxed::Box<T>
std::rc::Rc<T>
std::sync::Arc<T>
```

Standard library: allocator

```
std::string::String
std::vec::Vec<T, A = Global>
std::boxed::Box<T, A = Global>
std::rc::Rc<T, A = Global>
std::sync::Arc<T, A = Global>
```

Data structures: struct

```
struct Pose {
    x: f64,
    y: f64,
}
```

Data structures: enum

```
enum Mode {
    Idle,
    Move(Pose, f64),
}
```

Data structures: composition

```
struct Robot {
    name: String,
    pose: Pose,
    mode: Mode,
}
```

Generic Data Types

```
struct Transform<Scalar, Angle> {
    center: (Scalar, Scalar),
    angle: Angle,
    scale: Scalar,
}
```

std::result::Result

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

std::result::Result

```rust
enum Result<T, E> {
    Ok(T),
    Err(E),
}

type MyRet = Result<f64, f64>;
let x = MyRet::Ok(4.25);
let y = MyRet::Err(f64::NAN);

x.is_ok();              // true
x.is_err();             // false
x.map(|n| n * 2.0);     // Ok(8.5)
y.is_ok();              // false
y.is_err();             // true
y.map(|n| n * 2.0);     // Err(NAN)
```

Results: example

```
let file = File::create("data.txt");
match file {
    Err(e) => eprintln!("cant create file: {e:?}"),
    Ok(mut f) => {
        let ret = f.write_all(b"hello");
        match ret {
            Err(e) => eprintln!("cant create file: {
            Ok(()) => println!("file written."),
        }
    }
}
```

Results: shortcut

```
fn main() -> std::io::Result<()> {
    let mut file = File::create("data.txt")?;
    file.write_all(b"hello")?;
    println!("file written.");
    Ok(())
}
```

Results: shortcut

```rust
fn main() -> std::io::Result<()> {
    let mut file = File::create("data.txt")?;
    file.write_all(b"hello")?;
    println!("file written.");
    Ok(())
}

type std::io::Result<T> = Result<T, std::io::Error>;
```

std::option::Option

```
enum Option<T> {
    None,
    Some(T),
}
```

std::option::Option

```
enum Option<T> {
    None,
    Some(T),
}

type MyOpt = Option<f64>;
let x = MyOpt::Some(4.25);
let y = MyOpt::None;

x.is_some();            // true
x.is_none();            // false
x.map(|n| n * 2.0);     // Some(8.5)
y.is_some();            // false
y.is_none();            // true
y.map(|n| n * 2.0);     // None
```

Traits

```
trait SetSpeed {
    fn set_speed(&mut self, tgt: f64) -> Result;
}
```

Traits

```
trait SetSpeed {
    fn set_speed(&mut self, tgt: f64) -> Result;
}

impl SetSpeed for Robot {
    fn set_speed(&mut self, tgt: f64) -> Result {
        match self.mode {
            Mode::Idle => Err(Error::IsIdle),
            Mode::Move(pose, _) = {
                self.mode = Mode::Move(pose, tgt);
                Ok(())
            }
        }
    }
}
```

Async

```rust
#[tokio::main]
async fn main() -> Result {
    let port = 1234;
    tokio::spawn(async move {
        serve(port).await;
    });

    let mut client = Client::connect(port).await?;
    client.set("hello", "world").await?;

    let result = client.get("hello").await?;
    println!("got {result:?}");

    Ok(())
}
```

Channels

```rust
#[tokio::main]
async fn main() {
    let (tx, mut rx) = mpsc::channel(32);

    tokio::spawn(async move {
        tx.send("sending from first handle").await;
    });

    while let Some(message) = rx.recv().await {
        println!("GOT = {}", message);
    }
}
```

Rust Embedded

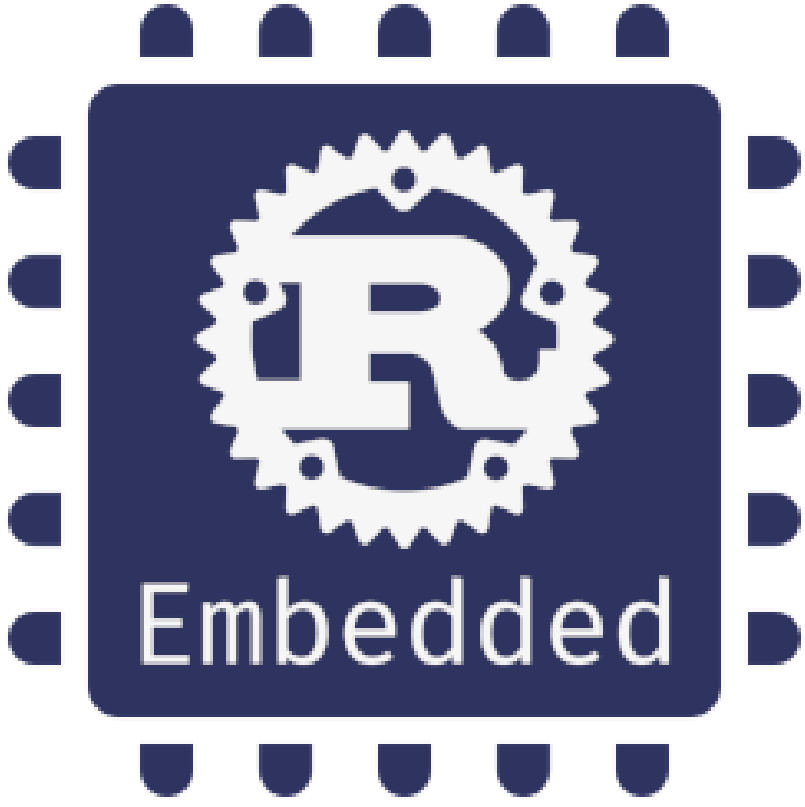

Figure 3: Rust Embedded

Ecosystem

- PAC: Peripheral Access Crates
- HAL: Hardware Abstraction Layers
- embedded-hal

Ecosystem

- PAC: Peripheral Access Crates
- HAL: Hardware Abstraction Layers
- embedded-hal

ref. https://www.youtube.com/watch?v=vLYit_HHPaY

Frameworks

- RTIC
- Embassy

Web

Leptos

Figure 4: Fast full-stack web apps in Rust

Tooling

LAAS CNRS

Clippy

```
$ rustup component add clippy
$ cargo clippy
$ cargo clippy --fix
```

espflash

```
$ cargo install probe-rs-tools
$ cargo embed
```

leptos

```
$ cargo install cargo-leptos
$ cargo leptos watch
```

Démo !

LAAS CNRS

toy example

https://github.com/nim65s/RobotS

Applications

LAAS CNRS

Shell utils

- grep → ripgrep
- find → fd
- ls → LSDeluxe / eza
- du → dust
- cat → bat
- ctrl-r → zoxide
- terminal → alacritty
- prompt → starship
- history → atuin
- make → just
- watch → watchexec / bacon
- screen / tmux → zellij
- diff → delta
- bash → nushell

Tools for other langages

Python

- ruff
- uv

JavaScript

- deno
- parcel

Alternatives

LAAS CNRS

Similar

- Go
- Swift

Even more modern

- Zig

Limitations

Learning curve

Not that easy

No shared objects

Recompile all the things

Interractions with outside world

You might need nix ;)

Questions ?

Thanks for your attention :)

References

- https://doc.rust-lang.org/rust-by-example/
- https://doc.rust-lang.org/book/