

@codecentric

Rust is easy

Things I would have
liked to know before
I'd started learning
Rust



Goetz Markgraf

Consultant codecentric AG

- goetz.markgraf@codecentric.de
- <https://www.linkedin.com/in/goetz-markgraf>



Goetz Markgraf

Senior Agile Consultant bei codecentric AG
#gernePerDu



Disclaimer

This is not a talk to learn Rust. It is a talk to understand two of its concepts.

Rust is powerful and reasonable to use

Rust makes "cents"

@codecentric

But Rust is so difficult ...



@codecentric

Rust is admired

For **9 years in a row**, Rust is voted as the **most admired language** on the Stackoverflow Developer Survey.

- <https://survey.stackoverflow.co/2025/technology#admired-and-desired>
- <https://survey.stackoverflow.co/2024/technology/#admired-and-desired>
- <https://survey.stackoverflow.co/2023/#technology-admired-and-desired>
- <https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted>
- <https://survey.stackoverflow.co/2021#technology-most-loved-dreaded-and-wanted>
- <https://survey.stackoverflow.co/2020#most-loved-dreaded-and-wanted>
- ...

What about AI?



@codecentric

The Rust tooling is perfect for AI

- standard command:
 - `cargo build`
 - `cargo run`
 - `cargo test`
 - ...
- standard project file system layout

AI can easily create a Rust project.

AI can easily understand a Rust project.

Rust is just different!

Specially in these points ...

@codecentric

Memory Management



@codecentric

Memory Management

Management of data objects that are



created at runtime



have a dynamic size,
which might change
over time



are not only temporary
(within one function)

Two Normal Strategies

1. Manual memory management in code

- `malloc()` & `free()`
- C & C++
- > 70 % of all security bugs are the result of wrong memory management (*Microsoft*)

2. Automatic memory management at runtime

- aka "Garbage Collection"
- Runtime penalty

Ownership and Borrows



Every object that is created is treated like a physical "thing".



It exists only once.



Only one owner can "hold" (**own**) it.

Each piece of data can only have one variable as an owner

Lifetime of `message`:

```
fn example() {  
    let message: String = String::from("Hello, world.");  
    println!("I say: {}", message)  
}
```

Move an object:

```
let message: String = String::from("Hello, world.");  
let message2: String = message;  
println!("I say: {}", message2);  
println!("I say: {}", message)
```

Rusts error messages are actually helpful

error[E0382]: borrow of moved value: `message`

--> [src/main.rs:12:27](#)

```
|  
9 | let message = String::from("Hello, world.");  
|      ----- move occurs because `message` has type `String`, which does not implement the `Copy` trait  
10 | let message2 = message;  
|      ----- value moved here  
11 | println!("I say: {}", message2);  
12 | println!("I say: {}", message)  
|      ^^^^^^^ value borrowed here after move
```

= **note:** this error originates in the macro ``$crate::format_args_nl`` which comes from the expansion of the macro ``println`` (in Nightly builds, run with `-Z macro-backtrace` for more info)

help: consider cloning the value if the performance cost is acceptable  [Apply fix](#)

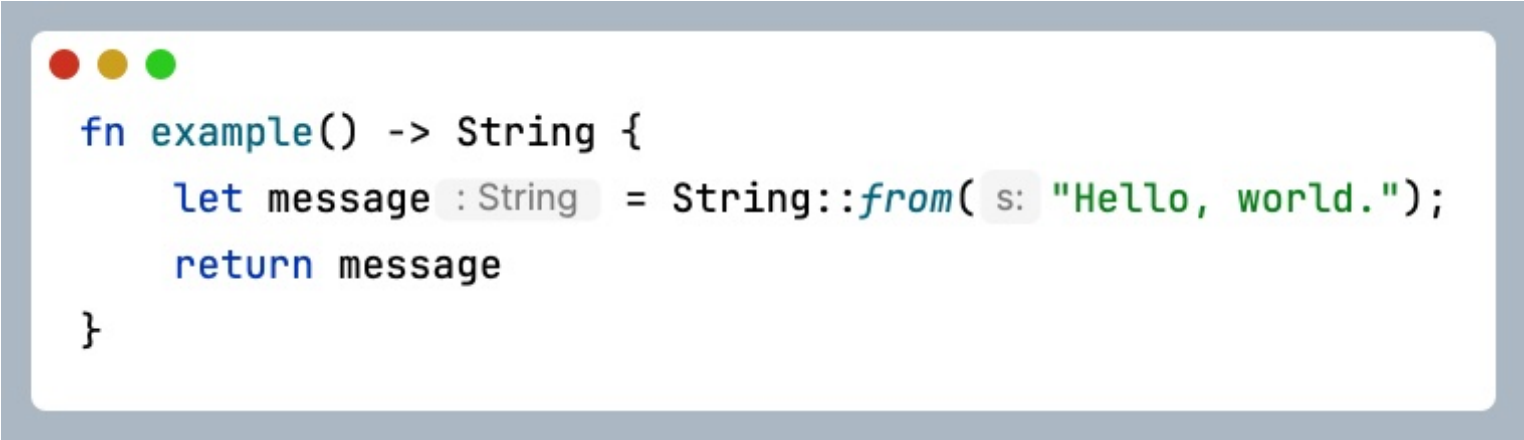
```
|  
10 | let message2 = message.clone();  
|      +++++++
```

An object can be passed to a function (as a parameter)



```
let message : String = String::from(s: "Hello, world.");  
some_function(message)
```

An object can be returned



```
fn example() -> String {  
    let message : String = String::from(s: "Hello, world.");  
    return message  
}
```

Wait, that can't be enough!



@codecentric

Let someone "look" at your object: references (borrows)

Simple rules:

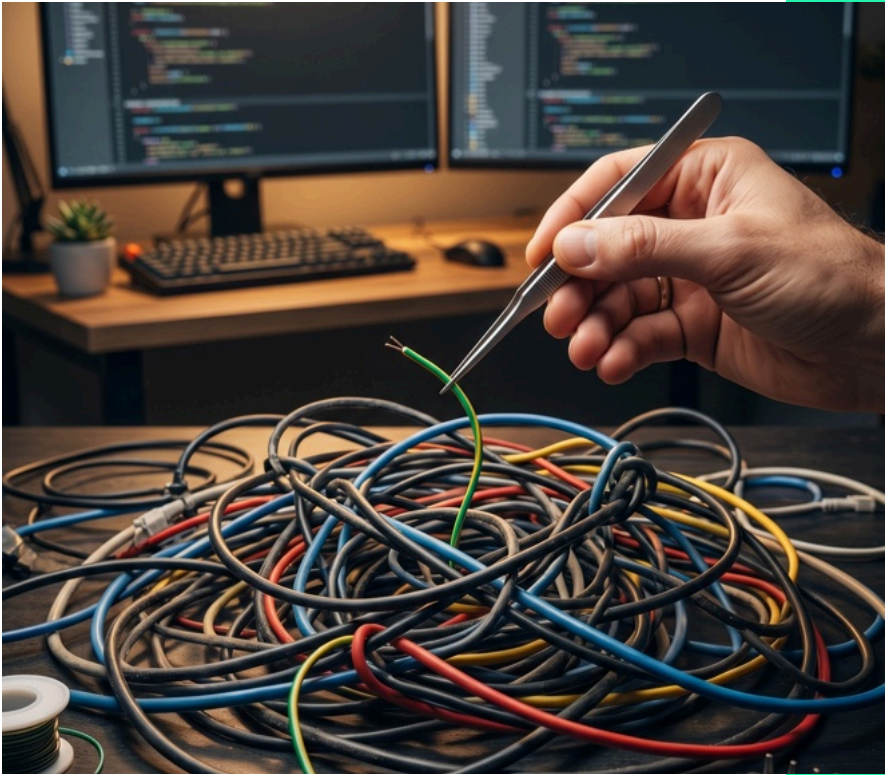
1. A reference is only valid as long as the object lives.
2. There can be **multiple read only** references
3. **XOR**: There can only be **one mutable** reference

And: The compiler has to be able to **track the lifetime** of an object and its references.

References go by &

```
fn example() {  
    let message : String = String::from(s: "Hello, world.");  
    let length : usize = string_length(&message);  
    println!("The string {} has length: {}", message, length)  
}
```

```
fn string_length(s: &String) -> usize {  
    s.len()  
}
```



Where it gets complicated

Example requirement: "get the third element from a list"

Get the third element from a list

- Remove the object from the list?
- Copy / Clone the object?
- Get a reference to the object?

```
fn get_third_from(list: Vec<String>) → String {}
```

```
fn get_third_from(list: &Vec<String>) → String {}
```

```
fn get_third_from(list: &mut Vec<String>) → String {}
```

```
fn get_third_from(list: &Vec<String>) → &String {}
```

```
fn get_third_from(list: Vec<&String>) → &String {}
```

The up-side

- "conscious programming"
- mutability without pain
- concurrent / async programming
- and: **no memory errors ...**

... with smaller footprint

Error Handling



@codecentric

This code is lying

```
fun readFileContent(path: Path): String {  
    if (!fileExists(path)) {  
        throw FileNotFoundException("File $path could not be found.")  
    }  
  
    return path.readText()  
}
```

rust never lies

```
fn read_file(file_path: &str) → Result<String, Error> {  
    read_to_string(file_path);  
}
```

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

Handle the error – consume directly

```
fn example() {  
    match read_file( file_path: "src/main.rs") {  
        Ok(content : String ) => println!("The content of the file is: {}", content),  
        Err(e : Error ) => println!("Error reading file: {}", e),  
    }  
}
```

Handle the error – default value



```
fn example() {  
    let content : String = read_file( file_path: "src/main.rs")  
    |  
    .unwrap_or(String::from( s: "Error reading file"));  
    println!("The content of the file is: {}", content);  
}
```

Handle the error – early return the error

```
fn example() -> Result<(), std::io::Error> {  
    let content : String = read_file( file_path: "src/main.rs" )?;  
    println!("The content of the file is: {}", content);  
    Ok::<(), std::io::Error>(())  
}
```

No null – no billion \$ mistake

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

The up-side

- "conscious programming"
- never forget to handle an error
- an application that is (nearly) **error free at runtime**

Like the Rustaceans say: *"... if it compiles it works"*

Are you hooked?



@codecentric

getting started

- Install everything with `rustup`.
- `cargo` is the build manager.
- IDE support:
 - IntelliJ RustRover
 - VS Code
 - NVim
 - zed
 - ...



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

Thank you



codecentric AG | Hochstraße 11 | D-42697 Solingen

@codecentric