

# **“Rust looks a lot better than other disasters” — a Rust introduction**

Meetup REWE Digital Ilmenau

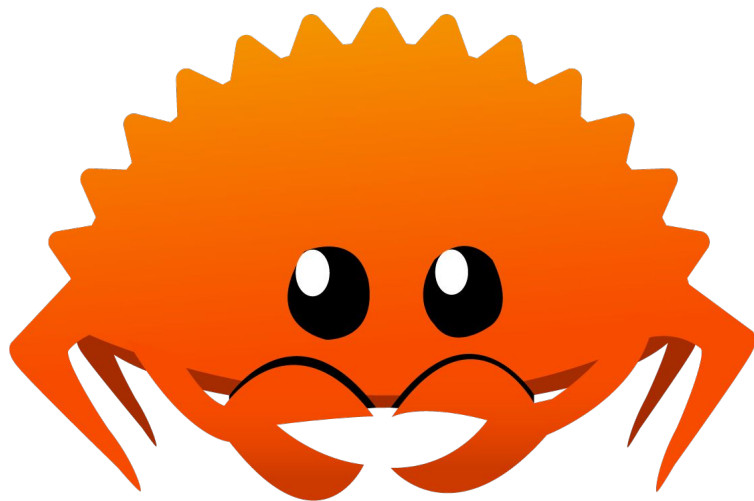
**REWE** digital

# Willkommen zu OCaml!



# Fun Fact

- First Rust Compiler written in OCaml (bootstrap)
- Rust does not have Bactrian camel (Trampeltier) but Ferris the Crab
- Ferris would have also been a wonderful title



# About me

Elmar Athmer

- 34 years old
- programming since 1998
- Living in Kassel
- Working at REWE Digital for 3 years
- I like video games (Civilization, yay)
- And running (HM PB 1:43:26), Full Marathon Berlin in 1,5 weeks :)



# Agenda

- 1) The Hype
- 2) Rust's promises
- 3) Safety vs Control
- 4) Rust basics
- 5) Ownership and Borrowing
- 6) Not covered
- 7) Outro & Questions

# “Rust looks a lot better than either of those two disasters”

Who said it?

- “We've had the [...] people who used Modula-2 or Ada, and I have to say Rust looks a lot better than either of *those* two disasters.”
- “I'm not convinced about Rust for an OS kernel [...] but at the same time there is no question that C has a lot of limitations”
- Do you remember...



Source: [Linux at 25: Linus Torvalds on the evolution and future of Linux](#)

**Sorry**



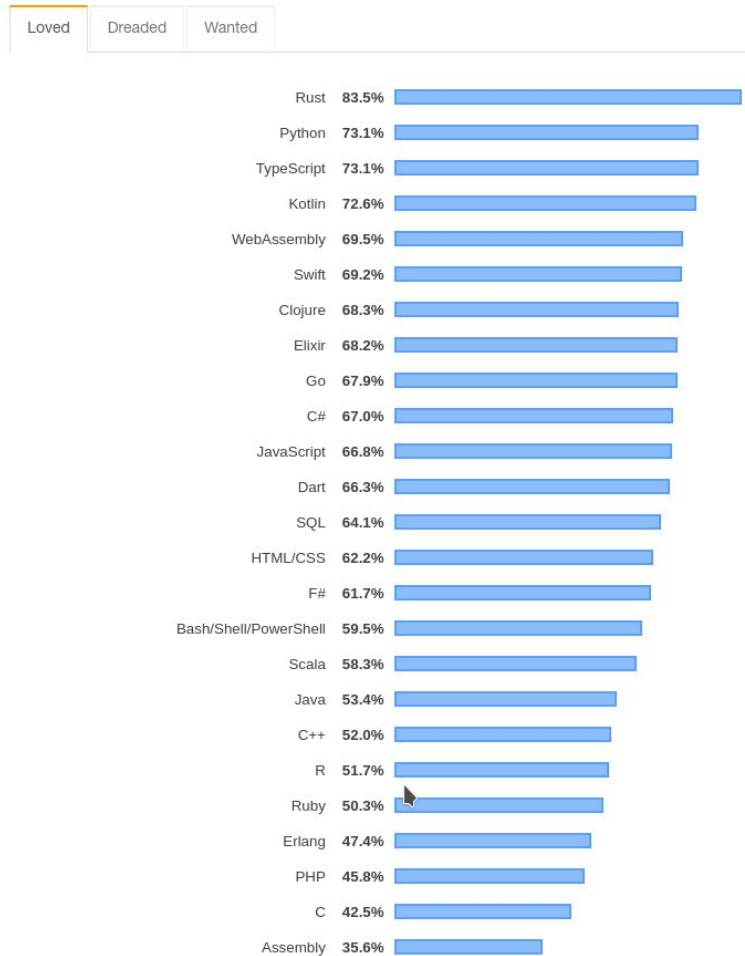
**For clickbait**

# What about other languages?

- “C++ is a horrible language.” — <http://harmful.cat-v.org/software/c++/linus>
- “I mean Java — I don’t care about it, what a horrible language” — <https://www.youtube.com/watch?v=Aa55RKWZxxI&t=44s>
- “The advantage of GC is that it is automatic. But GC apologists should just admit that it causes bad problems and often *encourages* people to write code that performs badly.” — <https://gcc.gnu.org/ml/gcc/2002-08/msg00552.html>



## Most Loved, Dreaded, and Wanted Languages



## What about non-Torvalds?

### Stack Overflow Developer Survey 2019

"For the fourth year in a row, Rust is the most loved programming language among our respondents."

<https://insights.stackoverflow.com/survey/2019#technology--most-loved-dreaded-and-wanted-languages>



## Those penguin-lunatics...

### Another quote

"Now we'll peek at why we think that Rust represents the best alternative to C and C++ currently available."

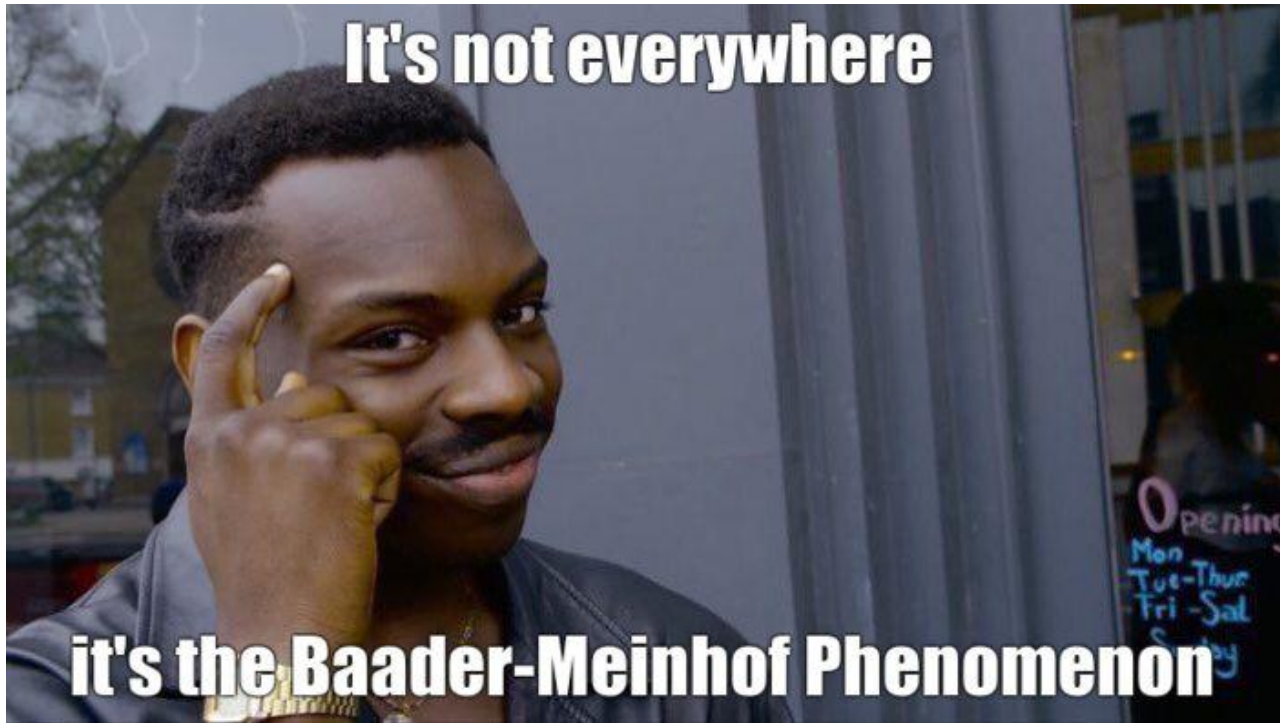
"This means that if that software had been written in Rust, 70% of these security issues would most likely have been eliminated."

<https://msrc-blog.microsoft.com/2019/07/22/why-rust-for-safe-systems-programming/>

**Rust, Rust, Rust**



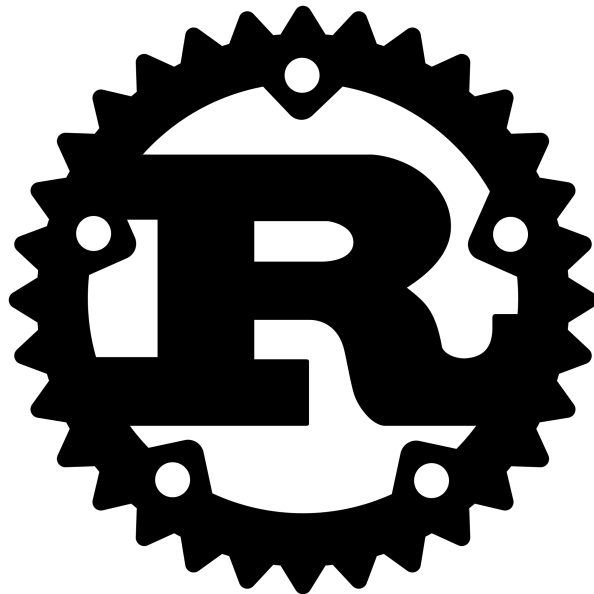
**Rust is everywhere**



[What's the Baader-Meinhof phenomenon](#)

# What is Rust?

- Developed by Mozilla since 2009/2010, [1.0 on 2015-05-15](#)
- Open Source, Multi-paradigm system programming language
- Provides memory safety
- Provides thread safety
- No Garbage Collection
  - No runtime overhead
  - Integrates with other Programming Languages (FFI)
- C++ like performance
- Strong, static typing



# *Multi-Paradigm, System programming*

Oh, and OpenSource of course

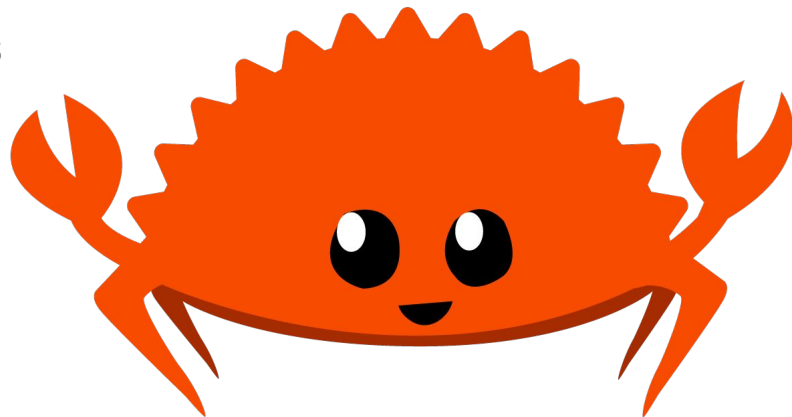
- Concurrent
- Functional ("passing functions as values", map, filter)
- Meta-Programming (Macros)
- Imperative (the "default")
- Object Oriented (should be familiar for most)

Get the buzzword checklist at [wikipedia](https://en.wikipedia.org/wiki/Buzzword_checklist).

# Multi-Paradigm, *System programming*

Oh, and OpenSource of course

- Operating Systems
- Software interacting closely with the operating systems
- Web Browsers (are more of a platform)
- In contrast to application development
- Typical examples are
  - C, C++
  - Maybe Go



**System programming?**

**Why should I care?**



**Now you CAN**

**do system programming**

# Side note

How much “performance” is necessary?

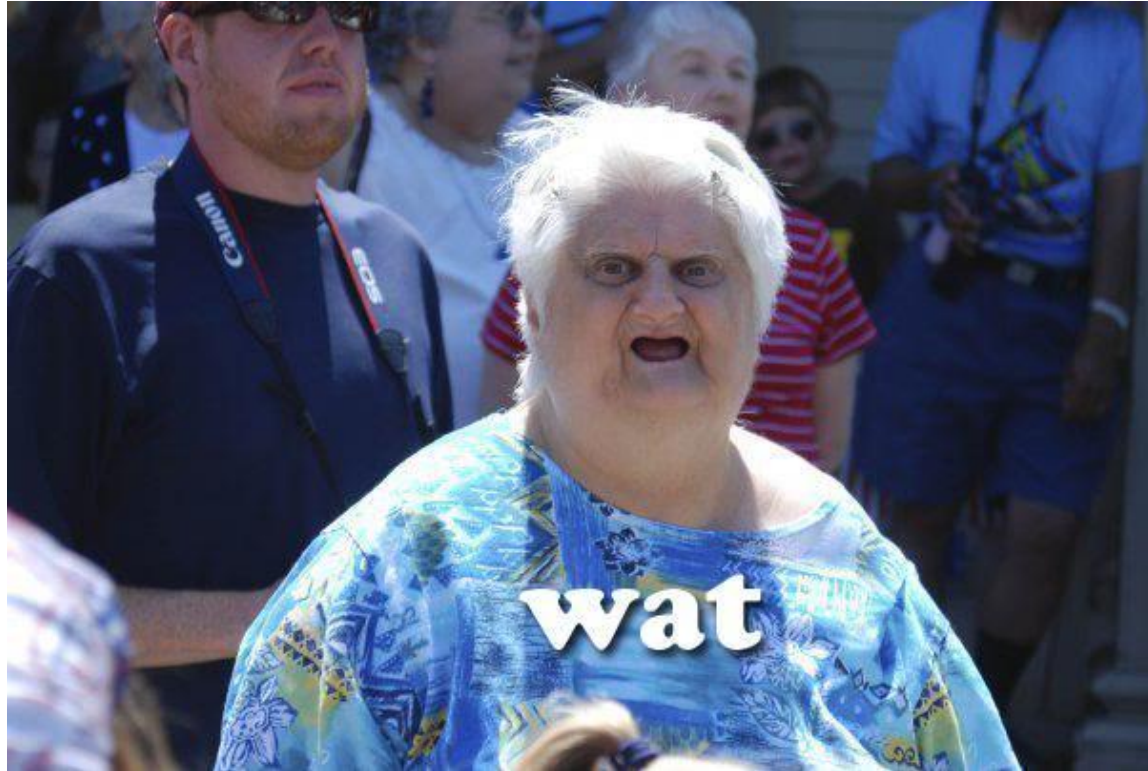
- General rule: a developer is more expensive than hardware
- IO is the real cost (that’s why NodeJS works)
- On the other hand — performance is not wrong
  - Developers spend much time NOT coding, so the relative cost for e.g. Rust overhead is small
  - It’s not “*worrying about efficiency in the wrong places [...]*”(premature optimization...)
- Code Rust like your high-level language (`map()`, `clone()`), your high-level language is just so wasteful with EVERY operation

# Sometime performance does matter

How much “performance” is necessary?

- *Sometimes*, costs are relevant: “a real cost saver” ([Zalando](#) about Rust with Kubernetes)
- *Sometimes*, performance is an enabler for new business logic/opportunities: [Deliveroo](#): “[...] dramatically sped up our dispatch process, and gave us more head-room in which we could try implementing more advanced algorithms.” (SIC)
- Functions as a service, Google Cloud Run etc.: startup, charged by exact execution time
- IOT, Raspberry Pi: Limited Resources (don’t forget battery)

# Memory safety, Thread safety?



# Use after free

No problem with GC



```
char* ptr = (char*)malloc (SIZE);  
...  
if (err) {  
    abrt = 1;  
    free(ptr);  
}  
...  
if (abrt) {  
    logError("operation aborted before commit", ptr);  
}
```

Source: [https://www.owasp.org/index.php/Using\\_freed\\_memory](https://www.owasp.org/index.php/Using_freed_memory)

# Double free

No problem with GC

- Confusion over which part of the program is responsible for freeing the memory

Source:

[https://www.owasp.org/index.php/Doubly\\_freeing\\_memory](https://www.owasp.org/index.php/Doubly_freeing_memory)



```
char* ptr = (char*)malloc (SIZE);
a(ptr);
b(ptr);
...
void a(char* ptr) {
    // do something with ptr
    free(ptr);
}
...
void b(char* ptr) {
    // do something with ptr
    free(ptr);
}
```

# Buffer overflow

Safe if programming language disallows direct memory access

- The buffer to store data overflows
- Nice (german) explanation:  
<https://www.heise.de/ct/artikel/Das-Sicherheitsloch-285320.html>

Source: [https://www.owasp.org/index.php/Buffer\\_Overflows](https://www.owasp.org/index.php/Buffer_Overflows)



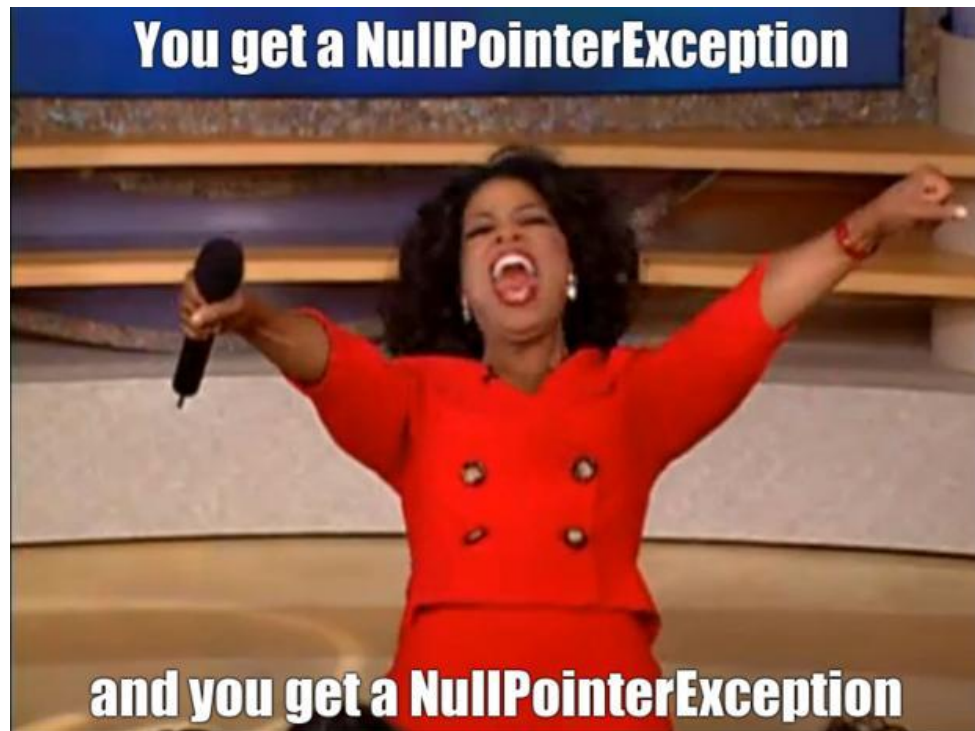
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void main(void) {
    char str[100] = scanf("%s");
    printf("%s", str);
}
```

# Null pointer dereference

Ever encountered one of these?

- NullPointerException
- "undefined is not a function"
- "can not read property 'x' of undefined"
- Aka "The billion dollar mistake"






# Data Race

Not to be confused with race condition

- two or more threads concurrently accessing a location of memory
- one of them is a write
- one of them is unsynchronized
- Also see <https://www.modernescpp.com/index.php/race-condition-versus-data-race>

Source:

<https://doc.rust-lang.org/nomicon/races.html>



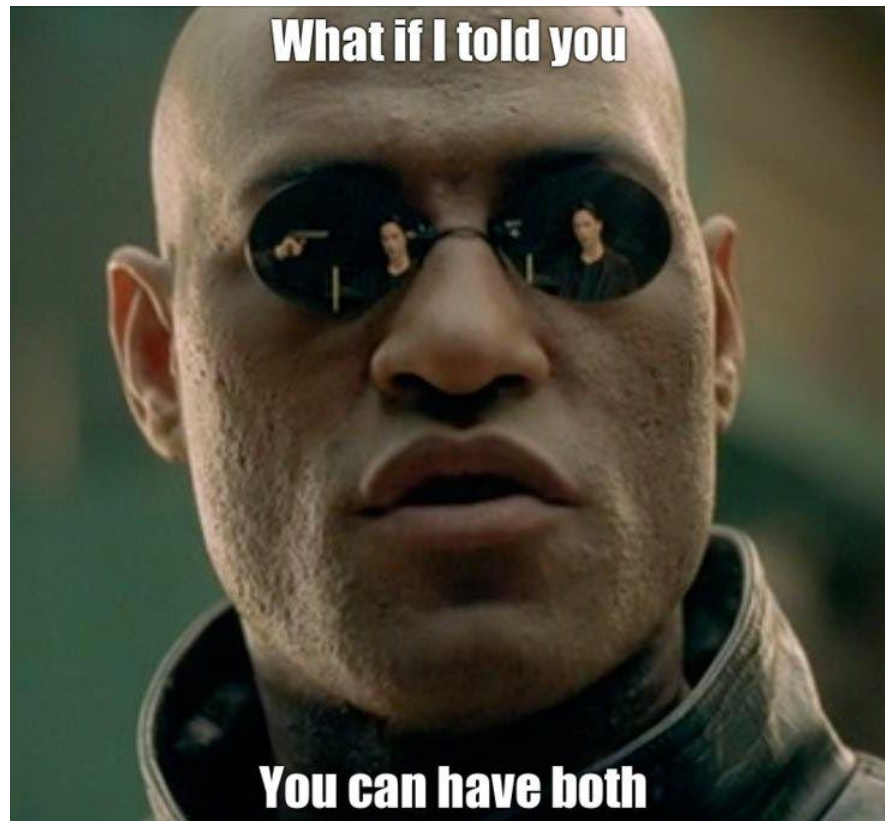
```
class BankAccount {
    int balance = 1000;

    pay_insurance() {
        new_balance = balance - insurance_amount;
        balance = new_balance;
    }

    pay_rent() {
        new_balance = balance - rent_amount;
        balance = new_balance;
    }

    start_of_month() {
        thread.spawn(pay_insurance);
        thread.spawn(pay_rent);
    }
}
```

# Safety vs Control



# Hello World

Let's have a look into Rust

- C like syntax
- Main is a simple function
- Type inference
- Variables defined with `let`



```
fn main() {  
    let name = "World";  
    println!("Hello {}", name);  
}
```

# Variables and datatypes

Nothing revolutionary here

- Immutable by default (**safety**)
- Explicit mutability with `mut` keyword
- Type annotation after variable name
  - Like TypeScript, Scala, etc.
- Lots of number types (`i32`, `i8`-`i128`, `usize` etc, remember: **control**)
- Vector as versatile collection type
- Tuples
- Full list

<https://doc.rust-lang.org/reference/types.html>



```
let mut counter: i32 = 0;  
counter = counter + 1;
```

```
let list: Vec<i32> = vec![1, 2, 3];
```

```
let coord = (5, 6);
```

# Null Pointer dereference protection

Must have for modern languages

- Option monad instead of null
- Known as/other solutions
  - [Maybe](#) in Haskell
  - [Optional](#) in Java
  - [Option](#) in Scala
  - [strictNullChecks](#) in TypeScript
- Option is an *Enum*

What else is happening here?

- [find\(\)](#) — functional
- Pattern matching
- [Shadowing](#) is idiomatic rust

```
enum Option<T> {  
    Some(T),  
    None,  
}  
  
fn sample() {  
    let l = vec![1, 2, 3];  
  
    let one: Option<i32> = l.iter().find(|&e| e == 1);  
    let one: Option<str> = one.map(|_o| "One");  
  
    match one {  
        None => println!("No one"),  
        Some(val) => println!("Found Some {}", val),  
    }  
}
```

# Error Handling — recoverable Errors

Checked exceptions done right

- Error monad instead of exceptions
- Errors encoded in Type
- Result is an *enum* (again)
- pattern matching (again)
- [Rust book on recoverable errors](#)

What else is happening here?

- Type Placeholder (Result<i32, \_>, Err(\_))
- Type annotation defines parse()'s return type

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

fn sample() {
    let input = "20 years";
    let age: Result<i32, _> = input.parse();

    match age {
        Ok(a) => println!("{}", years, a),
        Err(_) => println!("{}", ' is not a valid number", input),
    };
}
```

# Error Handling — unrecoverable Errors

panic!

- *Sometimes* you know better than the compiler: `unwrap()`
- Most of the time you don't
- This code panics: "thread 'main' panicked at 'This is a bug.: ParseIntError { kind: Overflow },'" — unrecoverable
- [Rust book on unrecoverable errors](#)



```
fn sample_unrecoverable() {  
    let localhost: IpAddr = "127.0.0.1".parse().unwrap();  
  
    let age: Result<i8, _> = "256".parse();  
    age.expect("This is a bug.");  
}
```

# The compiler is your friend

Rust has [...] a friendly compiler with useful error messages

- Type system catches lots of errors
- Getting code to compile can become cumbersome
- *"Compilation is the first unit test"* — Scott Hanselman (in any [hanselminutes](#) episode)
- If it compiles it works™





# Speaking of the compiler

Let's get down to business

- What about the other errors to catch at compile time?
  - Use after free
  - Double free
  - (memory leakage)
  - Data races
- **Garbage collection** is inefficient
- **Manual** memory management (malloc/free) is error prone



# Ownership & Borrowing

## Rust's USP

- Each value has a variable called it's *owner*
- Only **one** owner at a time
- When the owner goes out of scope, the value will be dropped
- Ownership can be *moved*
- After move, the value cannot be accessed anymore (could have been dropped already), prevents **use after free**

```
fn main() {  
    let a = String::from("my value");  
  
    // a moved to function  
    let b = do_something_with(a);  
  
    // b moved here  
    let _: () = do_something_else(b);  
  
    // err: value used here after move  
    do_something_with(b);  
}  
  
fn do_something_with(value_of_a: String) -> String {  
    // new value created here  
    let new_value = format!("given value is {} chars long", value_of_a);  
  
    // a dropped here  
  
    // this moves the new_value to a new owner (caller)  
    new_value  
}  
  
fn do_something_else(value_of_b: String) {  
    if b.contains(" ") {  
        println!("string contains spaces")  
    }  
  
    // value_of_b dropped here  
}
```

# Ownership & Borrowing

Sometimes you don't need to own a value

- By passing a *reference* the called function can *borrow* a value
- Borrows can be *mutable*, too (but only one borrow at a time)
- The **owning variable** needs to *live longer* than the **borrowed** variable

Did you notice?

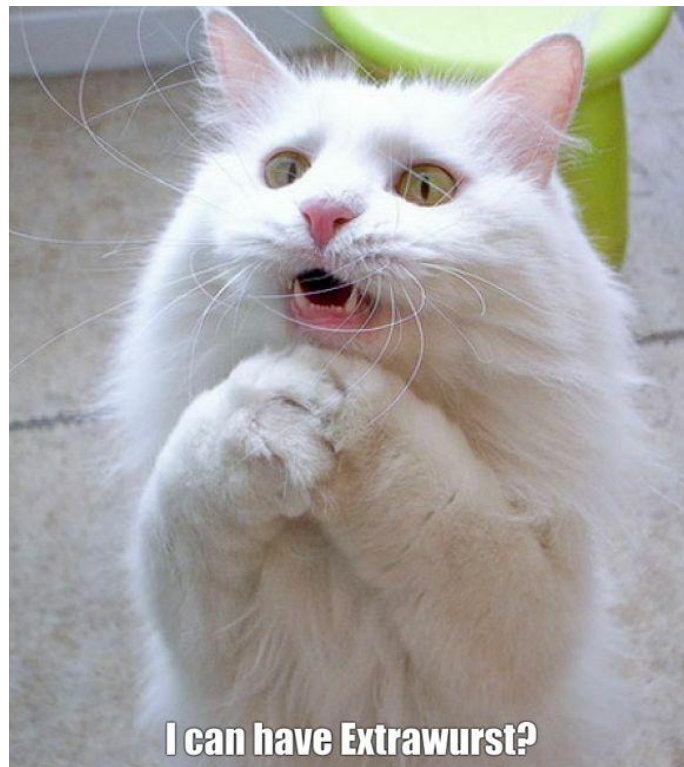
"Only one borrow at a time" prevents data races!

```
fn main() {  
    let a = String::from("my value");  
  
    let _len_of_a = calculate_length(&a);  
  
    let mut my_values = vec![&a];  
  
    let my_values_borrowed = &mut my_values;  
    // cannot borrow `my_values` as mutable more than once  
    // at a time  
    let my_values_borrowed2 = &mut my_values;  
  
    // borrow a mutable reference. Only one  
    add_a_value(my_values_borrowed)  
}  
  
fn calculate_length(s: &String) -> usize {  
    s.len()  
}  
  
fn add_a_value(list: &mut Vec<&String>) {  
    let new_value = String::from("my new value");  
    // borrowed value does not live long enough  
    list.push(&new_value);  
}
```

# Smart Pointers

When borrowing is not enough

- Owning/borrowing gets you pretty far, and is easy to understand so far
- `Rc<T>` for shared ownership — almost like Automatic Reference Counting for Objective-C/Swift
  - Problem: retain cycles
- And it's atomic counterparts `Arc<T>`
- Or `Mutex<T>` for concurrency
- `RefCell<T>` for internal mutability
- Out of scope...



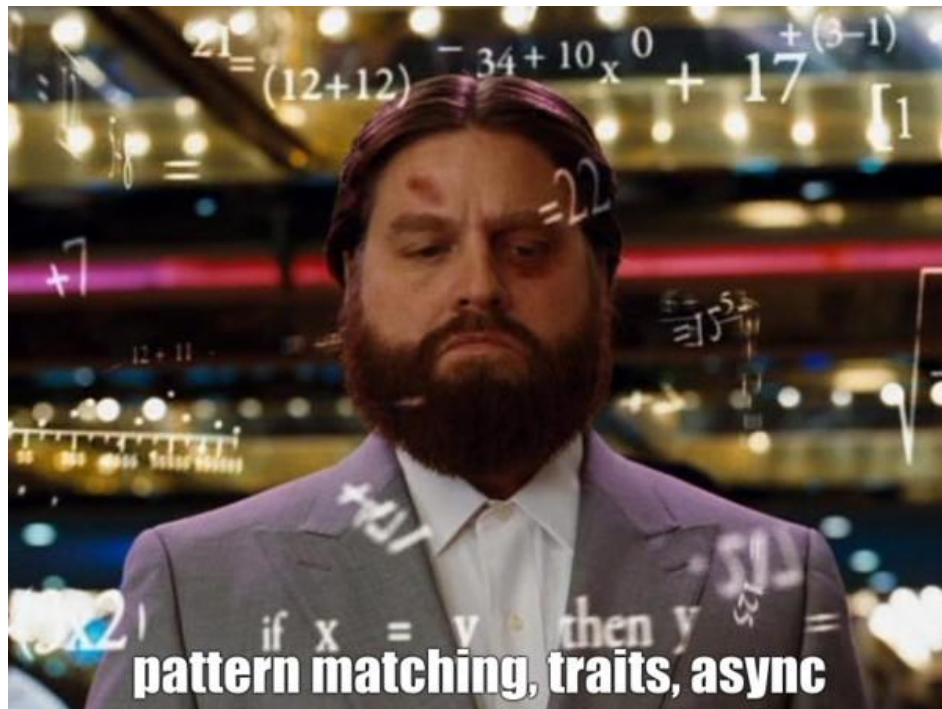
# OK, we covered the basics



# Language Features

We didn't cover...

- Enums, pattern matching
- Structs, Traits (OOP)
- Control flow
- Collections/Iterators
- Concurrency, Async/Await
- Unsafe rust
- macros



# Ecosystem

## Cargo and co

- Cargo — your one-stop-solution
- Cross compilation
- Integrated testing
- Package registry — [crates.io](https://crates.io)
- [Rustfmt](#)
- Linter — clippy
- [WebAssembly support](#)
- [Embedded support](#)
- Great development cycle
  - Releases every 6 weeks, backwards compatible
  - Editions for incompatible changes
  - unstable

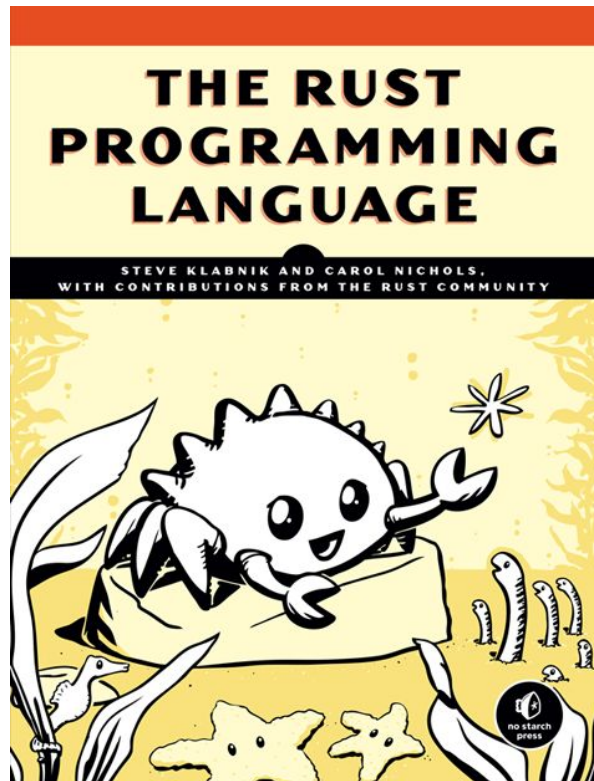


# Documentation

"Rust has great documentation"

- Built into cargo — `cargo doc`
- [The book](#)
- [Examples](#)
- [The Rust Standard Library](#)
- [Reference](#)
- [Documentation of all crates](#)

In general: <https://www.rust-lang.org/learn>





# Getting started

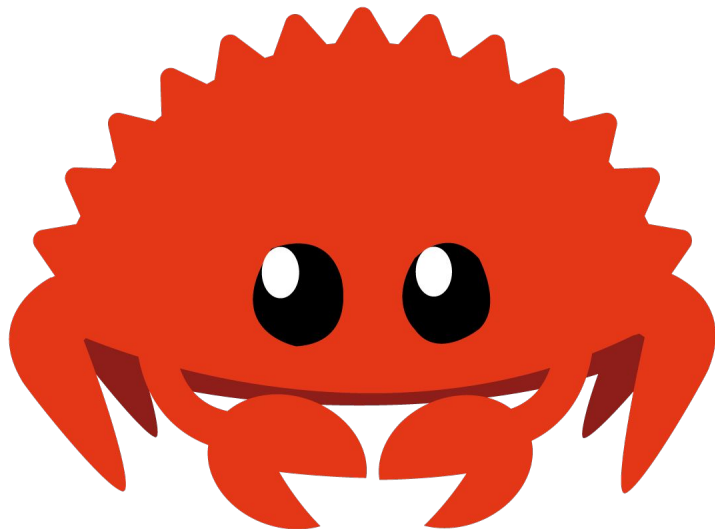
So, you want to learn rust?

- Read "the book"
- [The Website](#) is very useful: "[Get Started](#)", "[Install](#)", "[Learn](#)", "[Tools](#)"
- Solve some programming puzzles ([Project Euler](#), [Advent of code](#))
- Write a microservice
- Write native extensions for your language
- Program for your [Raspberry Pi in Rust](#)

# Additional resources

Food for hungry Rustaceans

- [Rust Language Cheat Sheet](#)
- Listen to the [“New Rustacean”](#) podcast (discontinued)
- Listen to [“Rustation Station”](#) podcast (unofficial successor)
- Read [This week in Rust](#)
- [The Rust programming language](#) — Google Tech Talk



A top-down photograph of various root vegetables arranged on a rustic, blue-painted wooden surface. The vegetables include several orange carrots of different sizes, some with green tops, several purple beets with green leaves, and a variety of potatoes in brown, red, and yellow skins. A bright pink rectangular box is centered over the image, containing the word "Questions?" in white, bold, sans-serif font.

**Questions?**





**Thank you very much!**





**Bonus slides**

# Other sources

If you can't get enough of rust

- “Rust is the future [...], C is the new Assembly”
  - <https://www.youtube.com/watch?v=I9hM0h6IQDo>

# Crates you should know

- [Rayon](#) — parallel iterators, putting all your Cores to work
- [Clap](#) — Command line argument parsing
- [Serde](#) and [serde\\_json](#) — serialization and deserialization
- [Diesel](#) — go to ORM/persistence
- [Actix Web](#) — current web framework with most momentum

# Success stories

- [Mozilla](#): Over the course of its lifetime, there have been 69 security bugs in Firefox's style component. If we'd had a time machine and could have written this component in Rust from the start, 51 (73.9%) of these bugs would not have been possible.
- [Zalando](#): "a real cost saver"
- [Npm](#): "npm chose Rust to handle CPU-bound bottlenecks."
- [Deliveroo](#): "moving from Ruby to Rust was a success that dramatically sped up our dispatch process, and gave us more head-room in which we could try implementing more advanced algorithms." [SIC]



# Things written in Rust

- Wireguard: [Wireguard rewrite](#), [Userspace wireguard](#)
- [HVVM](#): Migrating from OCaml
- [Facebook's Libra](#) may be controversial, but it's also written in Rust
- [Firefox' Quantum](#) and of course [servo](#)
- [Redox OS](#), a full operating system written in Rust
- Microsoft not only blogs about Rust, but [uses it](#) and [supports its usage](#)

# Integrating Rust with other languages

Perfect opportunity to sneak Rust into prod

- [Neon bindings](#) for NodeJS
- Native Ruby Extensions with [Helix](#)
- For python there is [PyO3](#)

# Linux kernel and Rust

- Framework for writing Linux kernel modules in safe Rust: [github: fishinabarrel/linux-kernel-module-rust](https://github.com/fishinabarrel/linux-kernel-module-rust)
- Rust driver framework could land upstream [lwn.net/Articles/797828/](https://lwn.net/Articles/797828/)