RUST EINFÜHRUNG
IN 10 MINUTEN

Alexander Korn

FOSS-AG Dortmund

INHALT

- Was ist Rust?
- Warum Rust?
- Nachteile?
- Hello World!
- Mutable?
- Referenzieren & Verleihen
- Unser erstes Projekt

WAS IST RUST?

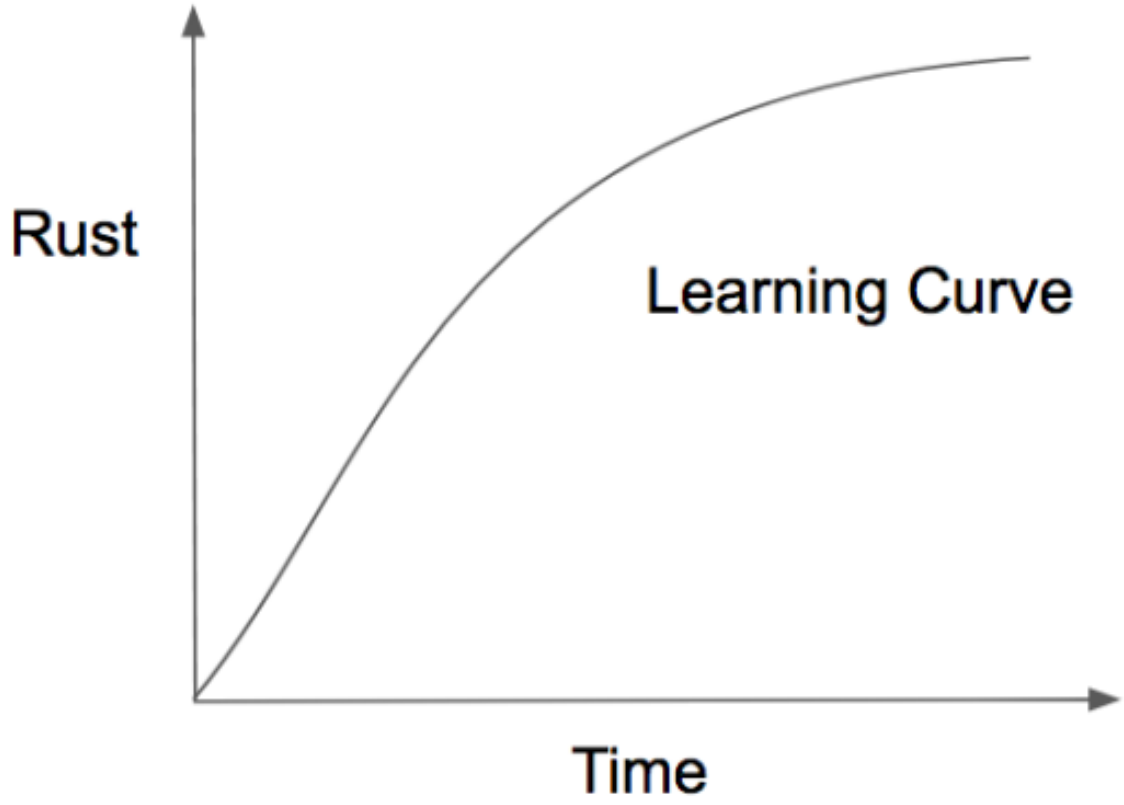
„Rust is a programming language that helps you write faster, more reliable software. High-level ergonomics and low-level control are often at odds with each other in programming language design; Rust stands to challenge that. Through balancing powerful technical capacity and a great developer experience, Rust gives you the option to control low-level details (such as memory usage) without all the hassle traditionally associated with such control."

- The Rust Programming Language (Abruf 18.04.2018)

WARUM RUST?

- **Speichersicherheit** — In kompilierbaren Programmen kann es keine Überläufe etc. geben
- **Schnelligkeit** — Rust ist genauso schnell, wie C/C++
- **Vielfältigkeit** — Man kann Rust für praktisch alles benutzen
- **Cargo** — Rust hat einen eingebauten Paket- und Projektmanager

NACHTEILE?
(UNTER ANDEREM)



Quelle: www.xtecher.com (Abruf 18.04.2018)

HELLO WORLD!

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

HELLO WORLD!

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

- `println!` ist ein **Makro**.
- Übersteigt den Inhalt dieses Vortrags.

MUTABLE?

Wir definieren eine Variable:

```
let x: String = String::from("hi");
```

Nun gilt:

- Die definierte Variable ist eigentlich eine **Konstante**, denn Sie lässt sich in Rust nicht verändern.
- Man nennt diese Definition „immutable“.
- Nebenbei bemerkt: Es gibt in Rust auch richtige Konstanten.

MUTABLE?

Wir definieren erneut eine Variable:

```
let mut x: String = String::from("hi");
```

Nun gilt:

- Dies ist eine **richtige** Variable, denn Sie wurde mit dem Stichwort `mut` definiert und lässt sich verändern.
- Man nennt diese Defintion „mutable“.

REFERENZIEREN & VERLEIHEN

Wir definieren eine Funktion:

```
fn output(i: String) {
    println!("{}", i);
}
```

Was macht die Funktion? → Sie gibt einen String aus.

REFERENZIEREN & VERLEIHEN

Wir definieren eine Funktion:

```
fn output(i: String) {
    println!("{}", i);
}
```

Was macht die Funktion? → Sie gibt einen String aus.

Wo ist das Problem? → Sie ergreift Besitz über den Wert, welcher ihr übergeben wird.

REFERENZIEREN & VERLEIHEN

Ein Beispiel - Wir definieren eine Variable und übergeben sie an unsere Funktion:

```
1  let x: String = …;   // Variable wird definiert
2
3  output(x);           // Funktion wird aufgerufen
4
5  println!("{}", x);   // Versuch, die Variable auszugeben
```

Wo ist das Problem? → Zeile 5 wird fehlschlagen, da der Besitz der Variable an die Funktion `add_one` übergeben wurde.
d. h., die Variable existiert im aktuellen Scope nicht mehr.

REFERENZIEREN & VERLEIHEN

Wie können wir das Problem lösen?

- Wir wollen nicht den Besitz der Variable, sondern eine Referenz weitergeben.
- Dazu ergänzen unsere Funktion und unseren Funktionsaufruf um ein Und-Zeichen.

```
fn output(i: &String) {              // Neue Funktion
    println!("{}", i);
}

output(&x);                          // Neuer Aufruf
```

UNSER ERSTES PROJEKT:
ERSTELLEN DES PROJEKTS

Wir nutzen Cargo, um ein neues Projekt zu erstellen:

```
$ cargo new guessing_game –bin
```

In unserem Projektordner finden wir nun die Datei `Cargo.toml`, in welche wir Informationen über uns Projekt eintragen:

```
1  [package]
2  name = "guessing_game"
3  version = "0.1.0"
4  authors = ["An author <an@author.de>"]
5
6  [dependencies]
   rand = "0.3.14"
```

Quelle (diese und folgende Folien): Ungefähr nach „The Rust Programming Language“ (Abruf 18.04.2018)

UNSER ERSTES PROJEKT:

NUTZEREINGABEN AKZEPTIEREN

```
use std::io;

fn main() {
    println!("Rate eine Zahl!");

    let mut guess = String::new();
    io::stdin().read_line(&mut guess)
        .expect("Konnte Zeile nicht lesen.");

    println!("Deine Eingabe: {}", guess);
}
```

UNSER ERSTES PROJEKT:

EINE ZUFALLSZAHL GENERIEREN

```
extern crate rand;

use std::io;
use rand::Rng;

fn main() {
    println!("Rate eine Zahl!");

    let secret = rand::thread_rng().gen_range(1, 101);

    let mut guess = String::new();
    io::stdin().read_line(&mut guess)
        .expect("Konnte Zeile nicht lesen.");

    println!("Deine Eingabe: {}", guess);
}
```

UNSER ERSTES PROJEKT:

DIE EINGABE MIT DER ZUFALLSZAHL VERGLEICHEN

```
extern crate rand;

use std::io;
use rand::Rng;

fn main() {
    println!("Rate eine Zahl!");

    let secret = rand::thread_rng().gen_range(1, 101);

    let mut guess = String::new();
    io::stdin().read_line(&mut guess)
        .expect("Konnte Zeile nicht lesen.");

    let guess: u32 = guess.trim().parse()
        .expect("Bitte gib eine Zahl ein!");

    println!("Deine Eingabe: {}", guess);

    match guess.cmp(&secret) {
        Ordering::Less    => println!("Zu klein!"),
        Ordering::Greater => println!("Zu groß!"),
        Ordering::Equal   => println!("Du hast gewonnen!"),
    }
}
```

UNSER ERSTES PROJEKT:

MEHRERE VERSUCHE MITTELS EINER SCHLEIFE

```
1   extern crate rand;
2
3   use std::io;
4   use rand::Rng;
5
6   fn main() {
7       println!("Rate eine Zahl!");
8
9       let secret = rand::thread_rng().gen_range(1, 101);
10
11      loop {
12          let mut guess = String::new();
13          io::stdin().read_line(&mut guess)
14              .expect("Konnte Zeile nicht lesen.");
15
16          let guess: u32 = guess.trim().parse()
17              .expect("Bitte gib eine Zahl ein!");
18
19          println!("Deine Eingabe: {}", guess);
20
21          match guess.cmp(&secret) {
22              Ordering::Less    => println!("Zu klein!"),
23              Ordering::Greater => println!("Zu groß!"),
24              Ordering::Equal   => {
25                  println!("Du hast gewonnen!");
26                  break;
27              },
28          }
29      }
30  }
```

UNSER ERSTES PROJEKT:

FEHLEINGABEN ABFANGEN

```
extern crate rand;

use std::io;
use rand::Rng;

fn main() {
    println!("Rate eine Zahl!");

    let secret = rand::thread_rng().gen_range(1, 101);

    loop {
        let mut guess = String::new();
        io::stdin().read_line(&mut guess)
            .expect("Konnte Zeile nicht lesen.");

        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_)  => continue,
        };

        println!("Deine Eingabe: {}", guess);

        match guess.cmp(&secret) {
            Ordering::Less    => println!("Zu klein!"),
            Ordering::Greater => println!("Zu groß!"),
            Ordering::Equal   => {
                println!("Du hast gewonnen!");
                break;
            },
}   }   }
```

UNSER ERSTES PROJEKT:
KOMPILIEREN & AUSFÜHREN

Wir kompilieren unser Projekt und führen es aus:

```
$ cargo run
   Compiling guessing_game v0.1.0 ...
Rate eine Zahl!
6
Deine Eingabe: 6
```

VIEL SPAß BEIM RUST LERNEN!

The Rust Programming Language Book Second Edition