Wer rastet, der rostet

Ein Rust-Kurs für Anfänger

Hendrik Wolff

Anton Obersteiner

Sommersemester 2023

Hört ihr Prog?

- Zielgruppe: 2tes Semester
- wir setzen Basics voraus
- sind wir zu schnell?
- oder ist euch langweilig?

⇒ redet mit uns

Voraussetzungen

- WSL / Linux / Dual Boot
- `rustup`: Verwaltet Rust-Compiler, -Packaging usw.
  - https://rustup.rs/
- `rustc` **der** Rust-Compiler
- `cargo` build system und package manager

Initiales toolchain setup mit rustup

```
curl -sSf https://sh.rustup.rs -o rustup.sh
sha256sum rustup.sh
# Ausgabe sollte sein (Stand 2023-04-20):
# 41262c98ae4effc2a752340608542d9fe411da73aca5fbe947fe45f61b7bd5cf
```

Initiales toolchain setup mit rustup

```
curl -sSf https://sh.rustup.rs -o rustup.sh
sha256sum rustup.sh
# Ausgabe sollte sein (Stand 2023-04-20):
# 41262c98ae4effc2a752340608542d9fe411da73aca5fbe947fe45f61b7bd5cf

less rustup.sh # ansehen, was das Skript ausführt
sh rustup.sh # starte Installation
```

Initiales toolchain setup mit rustup

```
curl -sSf https://sh.rustup.rs -o rustup.sh
sha256sum rustup.sh
# Ausgabe sollte sein (Stand 2023-04-20):
# 41262c98ae4effc2a752340608542d9fe411da73aca5fbe947fe45f61b7bd5cf

less rustup.sh # ansehen, was das Skript ausführt
sh rustup.sh # starte Installation

rustup default stable # set default toolchain
rustc --version
```

Editoren mit Rust-Support

GUI

- CLion + Rust plugin
- vscode / vscodium (+ rust_analyzer plugin)

Editoren mit Rust-Support

GUI

- CLion + Rust plugin
- vscode / vscodium (+ rust_analyzer plugin)

TUI

- helix
- vim / neovim
- emacs

Start a new project with cargo

- `cargo new lesson1` erstellt ein neues Rust Projekt im Ordner `lesson1`
- `cd lesson1`
- `cargo check` prüft Kompilerbarkeit
- `cargo build` kompiliert
- `cargo run` führt aus

Projektstruktur

Ordnerstruktur eines Rust Projektes:

```
├── Cargo.lock
├── Cargo.toml
├── src
│   └── main.rs
└── target
    ├── ...
```

entry point eines Rust Programms ist die `main` Funktion in der `src/main.rs` Datei:

```rust
// src/main.rs
fn main() {
    // your code goes here
}
```

Rust Tooling Overview

- `rustfmt` formatter
- `clippy` Linter (Annotiert euren Code beim Schreiben)
- `rustdoc` documentation builder
- `rust_analyzer` language server (für IDEs)

Helpful resources

- the Rust book
- rustlings excercises
- Rust by Example
- Rust Playground

rustc error messages

Der compiler ist dein Freund.

Er hilft dir.

Lies die error messages.

Skalare Typen

Rust	C	Beispiel
u8	char	b'A', 0b01000001
i8	signed char	-127
u128	long long unsigned	0xCOFFEE, 1_234_567_890
usize	unsigned	0usize
isize	int, ssize_t	12, -2
f32	float	let pi = 3.1415927f32;
f64	double	let pi: f64 = 3.141592653589793;
char	wchar	'A', 'ß'
bool	int, bool	true, false

Quiz 1: Typen

Maximaler Wert von `i32`?

Kompiliert das?

```
let pi = 3.1415f32;
let pi: f64 = 3.1415;
```

Quiz 1: Typen

Maximaler Wert von `i32`?

⇒ `4_294_967_295`

Kompiliert das?

```
let pi = 3.1415f32;
let pi: f64 = 3.1415;
```

Quiz 1: Typen

Maximaler Wert von `i32`?

⇒ `4_294_967_295`

Kompiliert das?

```
let pi = 3.1415f32;
let pi: f64 = 3.1415;
```

⇒ Ja

Compound Typen

```
let tuple: (i16, char, f32) = (-1_001, 'Σ', 21.593);
let empty_tuple: () = (); // is called a "unit"
```

Compound Typen

```rust
let tuple: (i16, char, f32) = (-1_001, 'Σ', 21.593);
let empty_tuple: () = (); // is called a "unit"

let array_a: [u64; 3] = [45, 101, 14_834_920];
println!("element on index 1 is: {}", array_a[1]);
```

Compound Typen

```rust
let tuple: (i16, char, f32) = (-1_001, 'Σ', 21.593);
let empty_tuple: () = (); // is called a "unit"

let array_a: [u64; 3] = [45, 101, 14_834_920];
println!("element on index 1 is: {}", array_a[1]);

let array_b = [12u8; 4];
let array_c: [u8; 4] = [12, 12, 12, 12];
assert_eq!(array_b, array_c);
```

Hilfreiche Macros

```
print!("write to stdout");
println!("write to stdout with newline");
```

Hilfreiche Macros

```
print!("write to stdout");
println!("write to stdout with newline");

println!("show me the value of my variable: {}", my_var);
println!("show me the value of my variable with debug formatting: {:?}", my_var);
println!("pretty print me the debug formatting of my variable: {:#?}", my_var);
```

Hilfreiche Macros

```
print!("write to stdout");
println!("write to stdout with newline");

println!("show me the value of my variable: {}", my_var);
println!("show me the value of my variable with debug formatting: {:?}", my_var);
println!("pretty print me the debug formatting of my variable: {:#?}", my_var);

eprint!("write to stderr");
eprintln!("write to stderr with newline");
```

Hilfreiche Macros

```
print!("write to stdout");
println!("write to stdout with newline");

println!("show me the value of my variable: {}", my_var);
println!("show me the value of my variable with debug formatting: {:?}", my_var);
println!("pretty print me the debug formatting of my variable: {:#?}", my_var);

eprint!("write to stderr");
eprintln!("write to stderr with newline");

dbg!("write file name, line number, variable names and contents to stderr");
```

Hilfreiche Macros

```rust
let a = 3;
let b = 4;

assert!(a == 3);
assert_eq!(a + 1, b); // eq ⇒ equal
assert_ne!(a,b); // ne ⇒ not equal
```

Variablen und Konstanten

- Variablen werden mit `let` erstellt
  snake_case
- Konstanten werden mit `const` erstellt
  SCREAMING_SNAKE_CASE

```
let tutor_count: u32;     // Deklaration
tutor_count = 4;          // Initialisierung
let student_count = 30;   // Deklaration & Initialisierung
const FIVE_MINUTES_IN_SECONDS: u32 = 5 * 60;
```

Variablen und Mutability

- Variablen sind immutable **by default**
  ⇒ Wert **kann nicht** verändert werden

```
let x = 5;
println!("The value of x is: {x}");
x = 6; // compile error
println!("The value of x is: {x}");
```

Variablen und Mutability

- mit **`mut`** keyword als mutable/veränderbar deklarieren
  ⇒ Wert **kann** verändert werden

```
let mut x = 5;
println!("The value of x is: {x}");
```

Variablen und Mutability

- mit **`mut`** keyword als mutable/veränderbar deklarieren
  ⇒ Wert **kann** verändert werden

```
let mut x = 5;
println!("The value of x is: {x}");

x = 6; // this works now
println!("The value of x is: {x}");
```

⇒ **explizite, optionale Veränderlichkeit**

Scope und Shadowing

- Scope = Gültigkeitsbereich für Variablen
- ein Scope beginnt mit { und endet mit }

```
// x is not valid here
fn main() { // start scope main function
    let x = 5; // x is now a valid variable
    println!("x is: {}", x); // ⇒ 5



} // end scope main function
// x is not valid here
```

Scope und Shadowing

- Scope = Gültigkeitsbereich für Variablen
- ein Scope beginnt mit { und endet mit }

```
// x is not valid here
fn main() { // start scope main function
    let x = 5; // x is now a valid variable
    println!("x is: {}", x); // ⇒ 5

    let x = x + 1; // shadowing x
    println!("x is: {}", x); // ⇒ 6

} // end scope main function
// x is not valid here
```

- Shadowing = Überschreiben einer Variable bis zum Ende des scopes
  ⇒ ausschließlich mit immutable Variablen arbeiten

Quiz 2: Scope und Shadowing

Was gibt das Programm aus?

```
let x = 5;
let x = x + 1;
{
    let x = x * 2;
    println!("value of x in inner scope: {x}");
}
println!("value of x: {x}");
```

Quiz 2: Scope und Shadowing

Was gibt das Programm aus?

```
let x = 5;
let x = x + 1;
{
    let x = x * 2;
    println!("value of x in inner scope: {x}");
}
println!("value of x: {x}");
```

```
value of x in inner scope: 12
value of x: 6
```

Mutability vs Shadowing

- Typ einer mutable Variable kann **nicht** verändert werden
- `mut` bezieht sich nur auf enthaltenen Wert

```
let mut spaces = "   ";
spaces = spaces.len(); // compile error
```

Mutability vs Shadowing

- Typ einer mutable Variable kann **nicht** verändert werden
- `mut` bezieht sich nur auf enthaltenen Wert

```
let mut spaces = "   ";
spaces = spaces.len(); // compile error
```

- Shadowing mit `let` deklariert neue Variable mit neuem Typen
  ⇒ gleichen Name wiederverwenden

```
let spaces = "   ";
let spaces = spaces.len(); // creates new variable
```

Exercise 1: Elementares Wissen über arrays

Erstelle ein Array mit mindestens 4 Elementen. Lies einen Index ein und gib das Element des Arrays an diesem Index aus. Nutze den folgenden Code, um den Index einzulesen.

Zusatz: Ersetze `usize` durch einen anderen Integer-Typen. Was passiert?

Gib verschiedene Indize ein und versuche vorherzusagen, was passiert.

```
let mut index = String::new();          //erstellt leeren String
std::io::stdin()                        //Standard-Input lesen: Termimal
    .read_line(&mut index)              //schreibt in den String index
    .expect("Failed to read line");     //Text für Fehler

let index: usize = index
    .trim()                             //Leerzeichen etc. weg
    .parse()                            //daraus usize machen
    .expect("Index entered was not a number");  //Text für Fehler
```