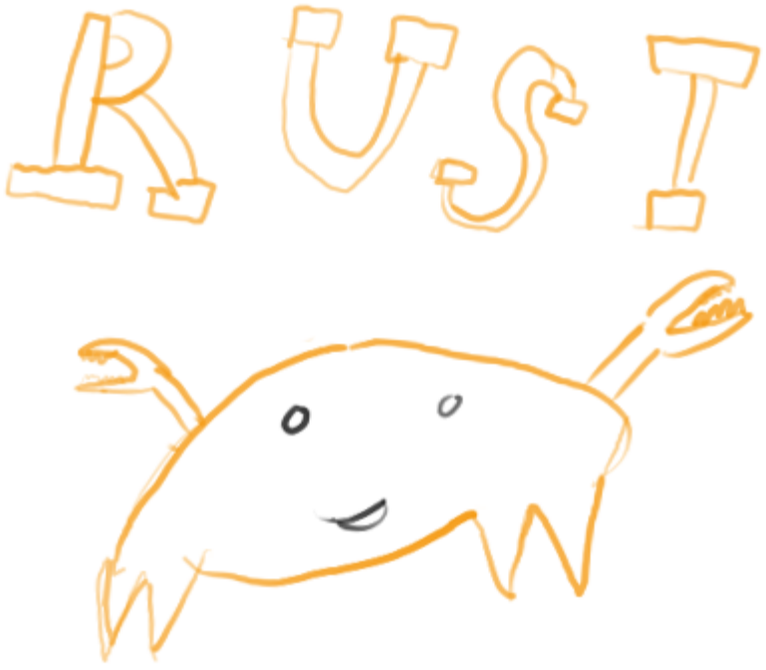
EINEN CLI MIT RUST ERSTELLEN



VORSTELLUNGSRUNDE

- Wie möchtet ihr Angesprochen werden?
- Wieviel könnt ihr schon programmieren?
- Habt ihr eine CLI Idee?

```rust
fn main() {
  let name = "David";
  println!("Hallo {name}");
}
```

```rust
fn main() {
  let name = "David";
  println!("Hallo {name}");
}
```

- *main* Funktionen wird ausgeführt
- *fn* für Funktionen
- Geschweifte Klammern

```
1 fn main() {
2   let name = "David";
3   println!("Hallo {name}");
4 }
```

- *let* für Variabelen
- Der Type wird meistens erkannt, selten muss man ihn angeben

```
1 fn main() {
2   let mut name = "David";
3   name = "Olivia";
4   println!("Hallo {name}");
5 }
```

- Unveränderbar außer man erlaubt veränderungen
- Mit *mut*

```rust
fn main() {
  let mut name = "David";
  name = "Olivia";
  println!("Hallo {name}");
}
```

```
$ cargo init
$ cargo run
```

Eigene Übung (ノ◕ワ◕)ノ

FUNKTIONEN

```
1 fn add(a: i32, b: i32) -> (i32, i32) {
2   (a + b, b)
3 }
4
5 let (sum, b) = add(2, 5);
6 //   7    5
```

IF-ANWEISUNGEN

IF-ANWEISUNGEN

```
1 if age >= 18 {
2   println!("Du bist volljährig");
3 } else {
4   println!("Du bist nicht volljährig");
5 }
```

IF-ANWEISUNGEN

```
let msg = if age >= 18 {
  "Du bist volljährig"
} else {
  "Du bist nicht volljährig"
};
```

ENUMS

ENUMS

```
enum Color {
  Red,
  Blue,
  Green
}
```

```
enum Color {
  Red,
  Blue,
  Green,
  Custom { r: u8, g: u8, b: u8 }
}
```

```rust
fn http_status() -> Result<u16, MyError> {}
fn shortend(str: &str) -> Option<char> {}
fn move_robot(len: i32) -> Result<(), RobotMoveError>
```

```
Ok(0)
Err(ColorErr::UnrecognizedFmt)

Some(12)
None
```

```
match color {
  Color::Red => println!("RGB(255, 0, 0)"),
  Color::Blue => println!("RGB(0, 0, 255)"),
  Color::Green => println!("RGB(0, 255, 0)"),
  Color::Custom { r, g, b } => println!("RGB({r}, {g}, {b})"),
}
```

```rust
use std::env;

match env::var("EDITOR") {
  Ok("nvim") => println!("Du nutzt Neovim!!!"),
  Ok(name) => println!("{}", name),
  Err(env::VarError::NotPresent) => println!("idk was du nutzt"),
  Err(env::VarError::NotUnicode(_)) => println!("Kein unicode"),
}
```

STRUCTS

STRUCTS

```
struct Person {
  name: String,
  age: u8,
  favourite_color: Color,
}
```

```rust
#[derive(Default, PartialEq, Eq)]
struct Person {
  name: String,
  age: u8,
  favourite_color: Color,
}

impl Person {
  fn new(name: String, age: u8, color: Color) -> Self {
    Person {
      name,
      age,
      favourite_color: color
    }
```

TRAITS

```
1 trait Summarize {
2   fn summarize(&self) -> String;
3 }
4
5 impl Summarize for Book {
6   fn summarize(&self) -> String {
7     String::from("hallo")
8   }
9 }
```

- Eigenes trait auf beliebigen type
- Fremdes trait auf eigenen type

CARGO

CARGO

```
$ cargo add clap -F derive
```

BORROW CHECKER

OWNERSHIP

- In Rust hat jeder Wert **genau einen** Owner
- Wenn der Owner out of scope geht dann wird er gelöscht

BORROWING

- Man kann beliebig viele Referenzen zu jedem Wert haben

```
 1 fn main() {
 2     let s1 = String::from("hello");
 3     let len = calculate_length(&s1);
 4     let len_2 = calculate_length(&s1);
 5
 6   dbg!(len);
 7   dbg!(len_2);
 8 }
 9
10 fn calculate_length(s: &String) -> usize {
11     s.len()
12 }
```

CLAP

CLAP

```rust
use clap::Parser;

#[derive(Parser)]
struct Cli {
  #[arg(short, long, default_value_t = true)]
  verify: bool,

  #[arg(short, long, default_value_t = false)]
  verbose: bool,
}

fn main() {
  let args = Cli::parse();
}
```

```rust
use clap::{Parser, Subcommand};

#[derive(Subcommand)]
enum Commands {
  Init,
  #[command(alias = "information")]
  Info { complete: Option<bool> }
}

#[derive(Parser)]
struct Cli {
  #[command(subcommands)]
  cmd: Commands,
  #[arg(short, long, default value t = false)]
```

HAPPY CODING