

**The Rust  
Programming  
Language**

**VEIT WEIDINGER**

Schweinfurt, 24.03.2021

# INHALTSVERZEICHNIS

1. Kurze Geschichte der Programmiersprachen
  - Von COBOL zu Rust
2. Warum Rust?
  - Mozilla Foundation
  - Einsatzgebiete
  - Paketmanager – Cargo
3. Spracheigenschaften
  - Variablen, Typen, IF-Else ...
4. Speicherverwaltung
  1. Ownership-System
  2. Zeiger Konzept
5. Zukunft der Sprache - Fazit



# GESCHICHTE

## VON COBOL ZU RUST

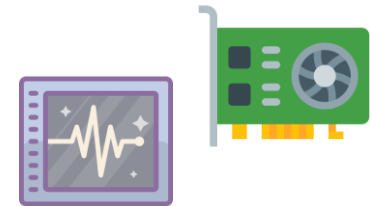
**COBOL**  
**1959**



**1972**



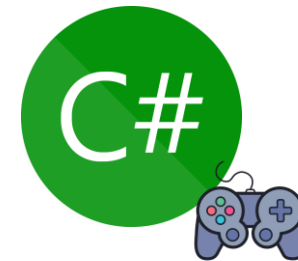
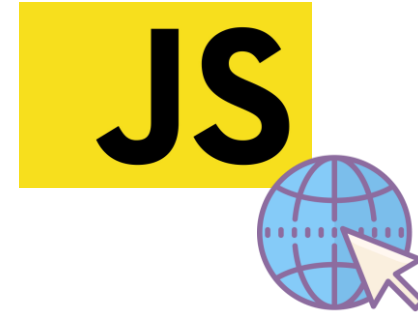
**1983**



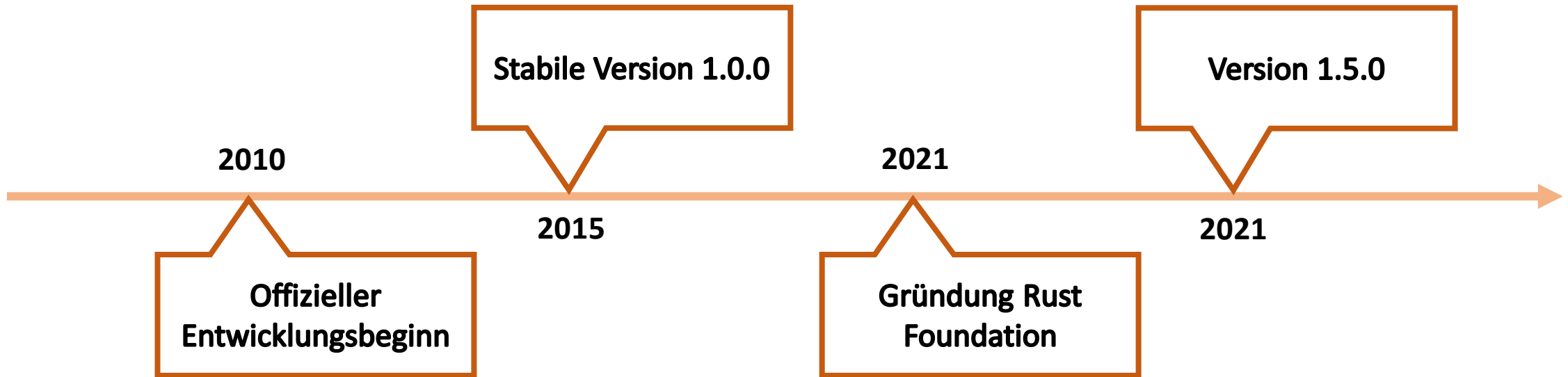
# GESCHICHTE VON COBOL ZU RUST



90er

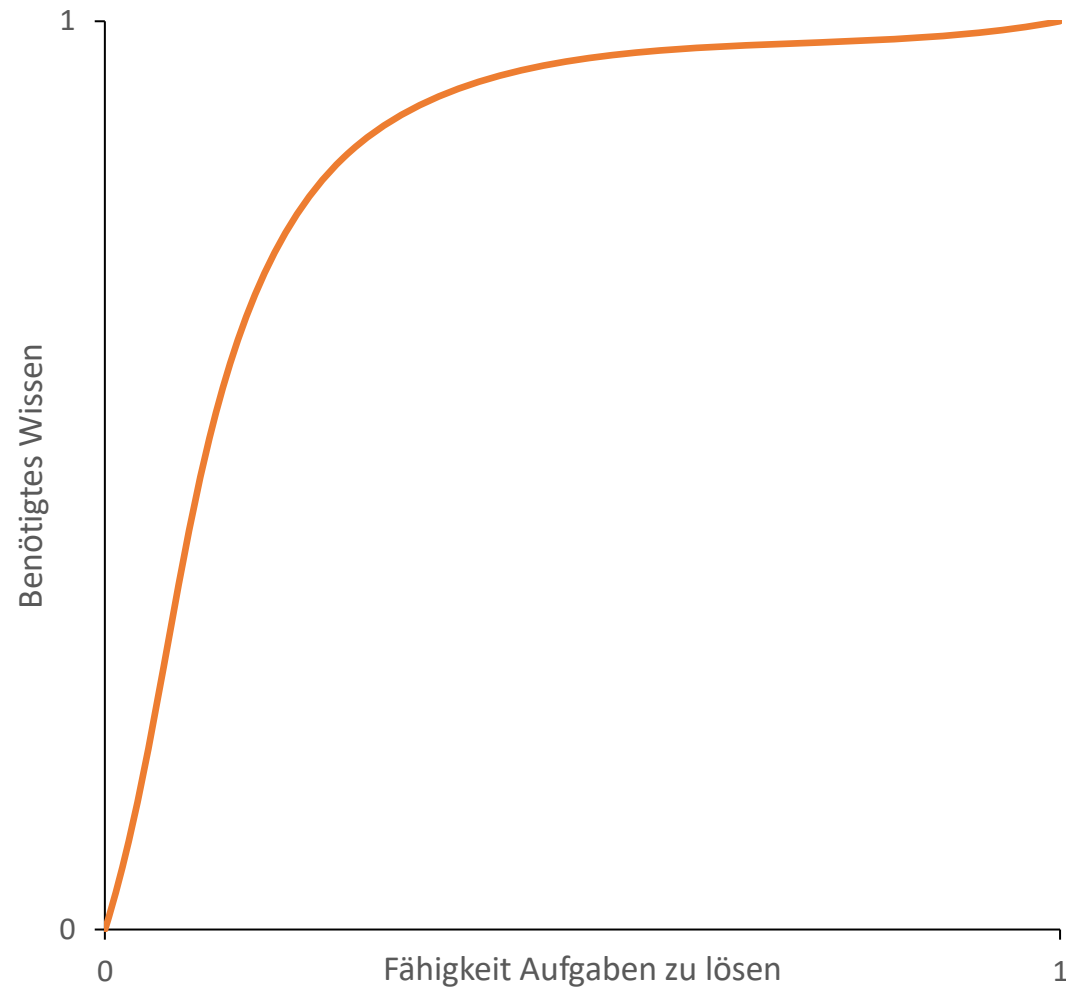


# GESCHICHTE RUST



# WARUM RUST?

## LERNKURVE



# WARUM RUST?

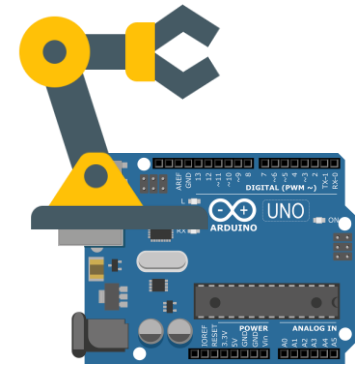
## EINSATZGEBIETE

Low- und  
High-Level  
Software  
entwickeln!

**moz://a**



Rust  
Foundation



# WARUM RUST?

## CARGO



<https://crates.io/>

```
[package]
name = "Fachreferat"
version = "0.1.0"
authors = ["Veit"]

[dependencies]
syn = "1.0.64"
```

Cargo.toml





# SPRACHEIGENSCHAFTEN

## VARIABLEN

```
let x = 7;  
let y = 2.22;  
let z = true;
```

```
x = 10; // Error: doppelte Zuweisung von immutable Variable „x“
```

```
let mut q = 4;  
q = 10; // passt
```

- Typinferenz: automatisch Typzuweisung
- Variablen werden mit **let** definiert und sind standardmäßig nicht änderbar (immutable)
- Erst durch den Zusatz **mut** werden sie änderbar (mutable)



# SPRACHEIGENSCHAFTEN

## TYPEN

### PRIMITIVE TYPEN

Integer

- **i8, i16, i32, i64**

Fließkommerzahlen

- float **f32**, double **f64**

Weitere:

- Boolean **bool** (**true** o. **false**)
- Unicode Char **char**
- String slice **str**

### TUPEL ( X, Y, ...)

- Beispiel:
  - ( **i8, bool** )
  - ( **char, bool, f64** )
- Zugriff mit **.0, .1, ...**

### ARRAY [TYP, Länge]

- Beispiel:
  - **[i8; 10]**
- Zugriff mit **[0], [1], ...**



# SPRACHEIGENSCHAFTEN

## TYPEN BEISPIELE

```
let int: i32 = 7;
let text: char = 'U';
let boolean: bool = true;

let tupel: (i8, bool) = (7, true);
let (first, second): (i8, bool) = (7, true); // destructuring
tupel.0 == first; // true
tupel.1 == second; // true

let abc: [char;5] = ['a', 'b', 'c', 'd', 'e'];
abc.len(); // --> 5
abc[0]; // --> a
abc[4]; // --> e
```



# SPRACHEIGENSCHAFTEN

## FUNKTIONEN

```
fn hello_world() {  
    println!("Hello World!")  
}  
  
fn print_summe(a: i32, b: i32) {  
    println!("Summe ist {}", a + b);  
}  
  
fn main() {  
    hello_world();  
    print_summe(21, 12); // Typinferenz  
}
```

- Erst Parametername und dann Typenbezeichnung
- „main“-Funktion ist der Einstiegspunkt in das Programm



# SPRACHEIGENSCHAFTEN

## FUNKTIONEN

```
fn summe(a: i32, b: i32) -> i32 {  
    a + b  
}  
  
fn teilen(a: i32, b: i32) -> i32 {  
    if b == 0 {  
        return 0; // early return  
    }  
    a / b  
}  
  
fn calc(a: i32, b: i32) -> (i32, i32) {  
    (a + b, a - b)  
}
```

- fn ... () -> Rückgabetype {...}
- Kein **return** nötig, aber möglich!
- Mehrere Werte zurückgeben mit **Tupel**



# SPRACHEIGENSCHAFTEN

## EXPRESSION ODER STATEMENT

### EXPRESSION

Geben **immer** einen Wert zurück

#### Literale

- 4 | "Hello World!" | false

#### Operationen

- 3 + 4 | 7 == 6

#### Funktionsaufrufe

- summe(3, 4) | calc(7, 2)

...

### STATEMENT

Geben **keinen** Wert zurück

- **let** Deklarationen
- Semikolon macht Expression zu Statement:
  - summe(3, 4);
  - 3 + 4;



# SPRACHEIGENSCHAFTEN

## IF-ELSE

```
let numb = 40;

if numb > 32 {
  // IF Zweig
} else if numb < 32 {
  // ELSE-IF Zweig"
} else {
  // ELSE
}

let result = if numb == 32 { 'U' } else { 'T' };

let result_2 = if numb == 32 {
  mach_was();
  7
} else {
  mach_was_anderes();
  'U' // Error: Typ unklar
};
```

- **Else** Zweig muss vorhanden sein
- **IF**, **ELSE-IF** und **ELSE** Zweig muss den gleichen TYP zurückgeben



# SPRACHEIGENSCHAFTEN

## SCHLEIFEN

```
while numb < 5 {  
    numb += 1;  
}  
  
loop { // unendlose Schleife  
    numb += 1; // --> Error: Integer überläuft  
}  
  
for i in 1..5 {  
    println!("{}", i);  
}  
  
let arr = [5, 11, 22];  
for c in &arr {  
    println!("{}", c);  
}
```

### **RANGE** ( start..ende )

- 1..5 -> 1,2,3,4
- 1..=5 -> 1,2,3,4,5

- **for**-Schleife ist performanter als **while**-Schleife

- **SYNTAX:**  
**for** variable\_name **in** expression { ... }





# SPRACHEIGENSCHAFTEN

## STYLE GUIDELINES



[Offizielle Style Guidelines](#)

### KOMMENTARE

```
// einzeilige Kommentare
```

```
// mehrzeilige Kommentare
```

```
// es gibt kein /* */
```

```
/// Kommentare zu Dokumentationszwecken
```

### NAMENSVERGABE

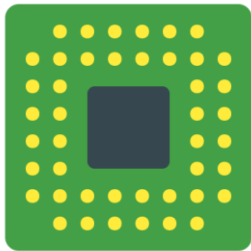
- **snake\_case**
  - Variablen, Funktionen
- **UpperCamelCase**
  - Typen
- **SCREAMING\_SNAKE\_CASE**
  - Konstanten



# SPEICHERVERWALTUNG

## ALLGEMEIN

Befehlssatz: x86



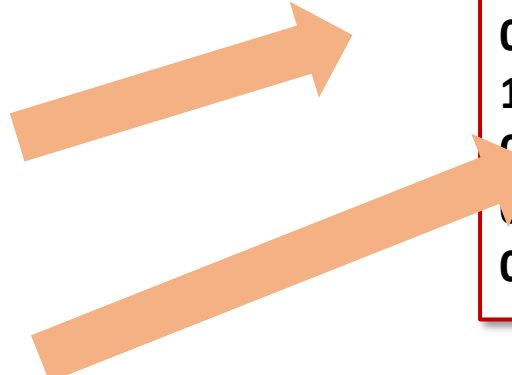
Variable Name: **zahl**

Zeiger: **010110** (64 Bit)

Variable Name: **zahl\_copy**

Zeiger: **010111** (64 Bit)

zahl2 = zahl;



```
00101001010111000111111111110000001
010100100101010010100100111100010100
010001000000100000000100010101001010
010100001010010000100011001010100100
110100001000100010010000100010010001
001001111000110000010000001000010010
000001010000101010010100000100000000
0100111111111111111100000000000011010
```

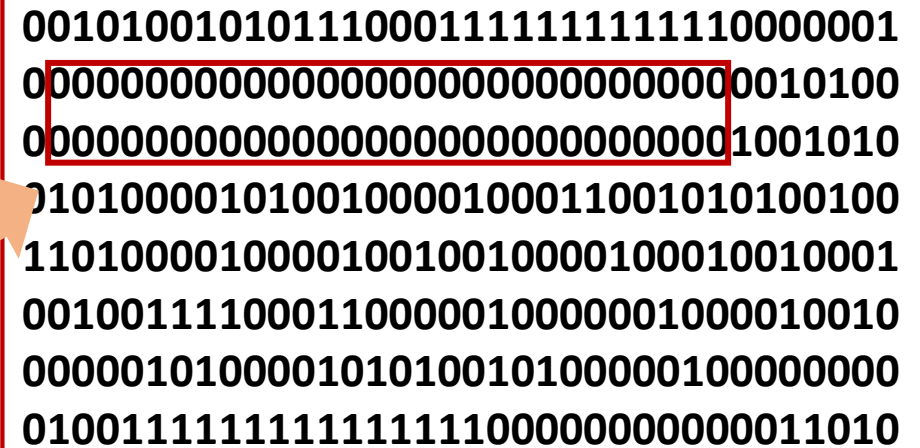
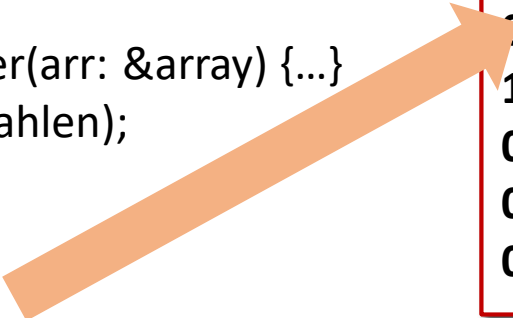
# SPEICHERVERWALTUNG

## ALLGEMEIN

Variable: **prim\_zahlen** (Array) ✗  
Zeiger: **011000** (64 Bit)

```
funktion get_random_number(arr: &array) {...}  
get_random_number(prim_zahlen);
```

Variable: arr  
Zeiger: **011001** (64 Bit)



001010010101110001111111111100000001  
0000000000000000000000000000000000010100  
000000000000000000000000000000000001001010  
010100001010010000100011001010100100  
110100001000010010010000100010010001  
001001111000110000010000001000010010  
000001010000101010010100000100000000  
010011111111111111100000000000011010



# SPEICHERVERWALTUNG

## OWNERSHIP-SYSTEM

### MOVE SEMANTICS (CLONE-TYPEN)

```
let a: String = "hi".to_string();
println!("a: {}", a);

let b = a;
println!("a: {}", a); // Error: Use of moved value!
```

```
let rosi = "Rosi".to_string();

fn greet(name: String) {
    println!("Hello: {}", name);
}

fn say_goodbye(name: String) {
    println!("Goodbye: {}", name);
}

greet(rosi);
say_goodbye(rosi); //Error: Use of moved value!
```

- Move überträgt **OWNERSHIP**
  - Variable die **moved** wurde ist danach nicht mehr verwendbar
- **OWNER** bestimmt über Lebenszeit des Wertes (explizite Laufzeit Angabe möglich, [hier mehr](#))



# SPEICHERVERWALTUNG

## BORROWING

```
let rosi = "Rosi".to_string();

fn greet(name: &String) {
    println!("Hello: {}", name);
}

fn say_goodbye(name: &String) {
    println!("Goodbye: {}", name);
}

greet(&rosi);
say_goodbye(&rosi);
```

```
let mut rosi = "Rosi".to_string();

fn greet(name: &mut String) {
    println!("Hello: {}", name);
    *name = "Hansi".to_string();
}

fn say_goodbye(name: &String) {
    println!("Goodbye: {}", name);
}

greet(&mut rosi);
say_goodbye(&rosi);
```

**Gleichzeitig** kann ein Wert entweder  
**viele immutable Borrows (Aliasing)** oder **ein mutable Borrow (Mutability)**  
besitzen



### Rust landet erstmals in wichtigem Linux-Kernel-Zweig

Die Sprache [Rust](#) ist erstmals in den [Linux-Kernel](#) aufgenommen worden. Bis zur Veröffentlichung im Hauptzweig steht aber noch viel Arbeit bevor.



22. März 2021, 10:41 Uhr, Sebastian Grüner



# FAZIT



**SAU SCHNELL!**



**MÄCHTIG!**



# FRAGEN

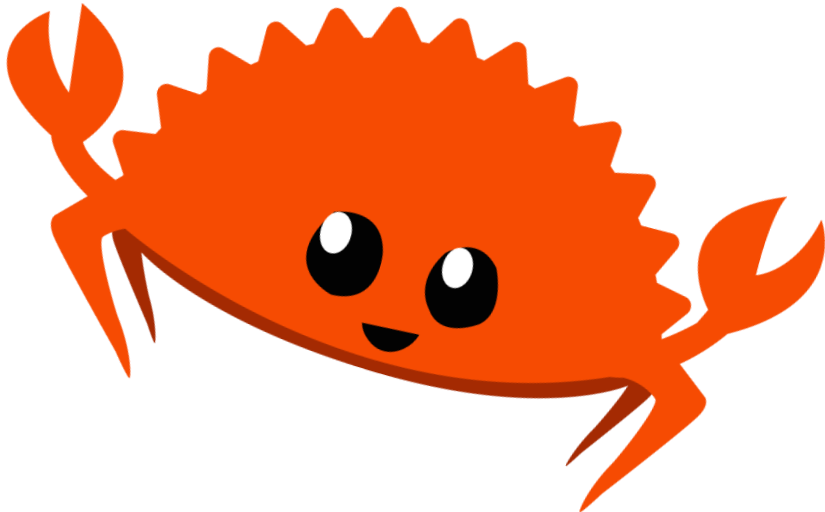




# HACK IT

## AUFGABEN

- Aufgaben unter <https://github.com/VWeidinger/rust-fosbos-sw>
- Rust Playground um Code zu testen: <https://play.rust-lang.org/>
- Beispielcode: <https://doc.rust-lang.org/stable/rust-by-example/> oder Handout/Foliensatz in [GitHub](#)



# QUELLEN

## INHALT

- <https://github.com/LukasKalbertodt/programmieren-in-rust/blob/master/slides/01-Grundlagen.pdf>
- <https://github.com/LukasKalbertodt/programmieren-in-rust/blob/master/slides/03-Ownership-System.pdf>
- <https://doc.rust-lang.org/stable/rust-by-example/>
- <https://doc.rust-lang.org/stable/book/> → Code Beispiele und Implementierungshilfen
- [https://en.wikipedia.org/wiki/Rust\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Rust_(programming_language))  
<https://stackoverflow.com/questions/580292/what-languages-are-windows-mac-os-x-and-linux-written-in>
- Gründungszahlen aus entsprechenden Wikipedia Seiten zu der Programmiersprache

## BILDER UND GRAFIKEN

- Icons: <https://icons8.de/>
- <https://forum.golem.de/staticrl/images/logo-g.png>
- <https://www.rust-lang.org/static/images/rust-social-wide.jpg>
- [https://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/Rust\\_programming\\_language\\_black\\_logo.svg/1200px-Rust\\_programming\\_language\\_black\\_logo.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/Rust_programming_language_black_logo.svg/1200px-Rust_programming_language_black_logo.svg.png)
- <https://crates.io/assets/Cargo-Logo-Small-c39abeb466d747f3be442698662c5260.png>
- <https://www.mozilla.org/media/protocol/img/logos/mozilla/black.40d1af88c248.svg>
- <https://www.mozilla.org/media/img/firefox/template/page-image-master.1b6efe3d5631.jpg>
- <https://rustacean.net/assets/rustacean-flat-happy.png>

