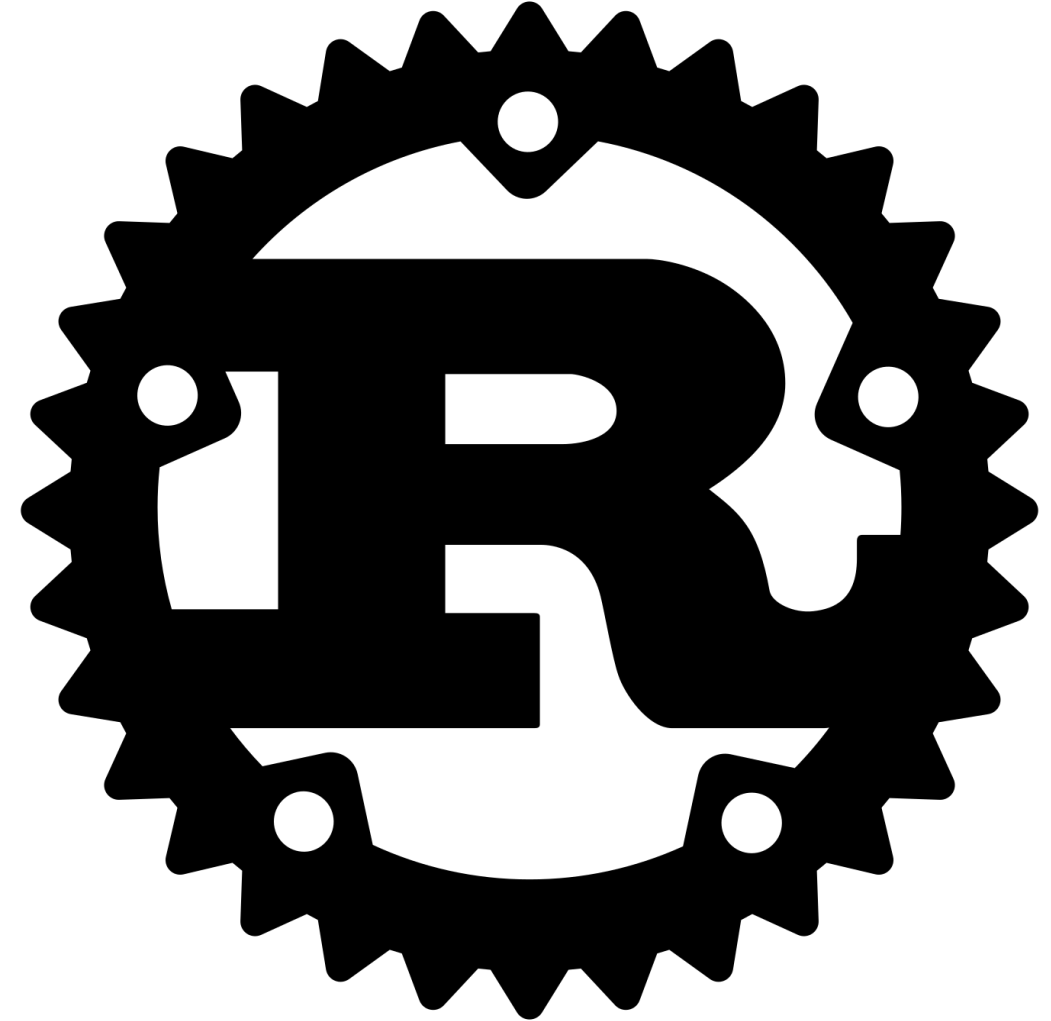


Rust - Workshop



Was machen wir?

- Überblick über das Rust Projekt ~ **10 Minuten**
- Einführung in Rust Grundlagen ~ **25 Minuten**
 - Vorstellung des Package Managers
 - Programmierkonzepte (Variablen & Mutierbarkeit, Datentypen, Funktionen...)
- Erste Aufgaben zum warm werden ~ **30 Minuten**
- **Pause** ~ **10 Minuten**
- Ownership Konzept – Memory Safety ~ **20-25 Minuten**
- Actix-Web Intro ~ **10 Minuten**
- Aufgabe: Web API mit Actix-Web ~ **15 Minuten**
- Zusammenfassung und Ausblick ~ **5 Minuten**

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Quelle: <https://www.rust-lang.org/>

Warum wurde das Rust Project ins Leben gerufen?

Ziel des Projekts:

→ Entwicklung einer **sicheren**, **nebenläufigen** und **praktischen** Systemprogrammiersprache

Hauptgründe für Rust:

Andere Sprachen bieten nicht genügend:

- Sicherheit
- Support für Nebenläufigkeit
- Praktikabilität
- Ressourcenkontrolle

Quelle: <https://prev.rust-lang.org/en-US/faq.html#what-is-this-projects-goal>

Wer steht hinter dem Projekt?

Platinum



Gold



Silver

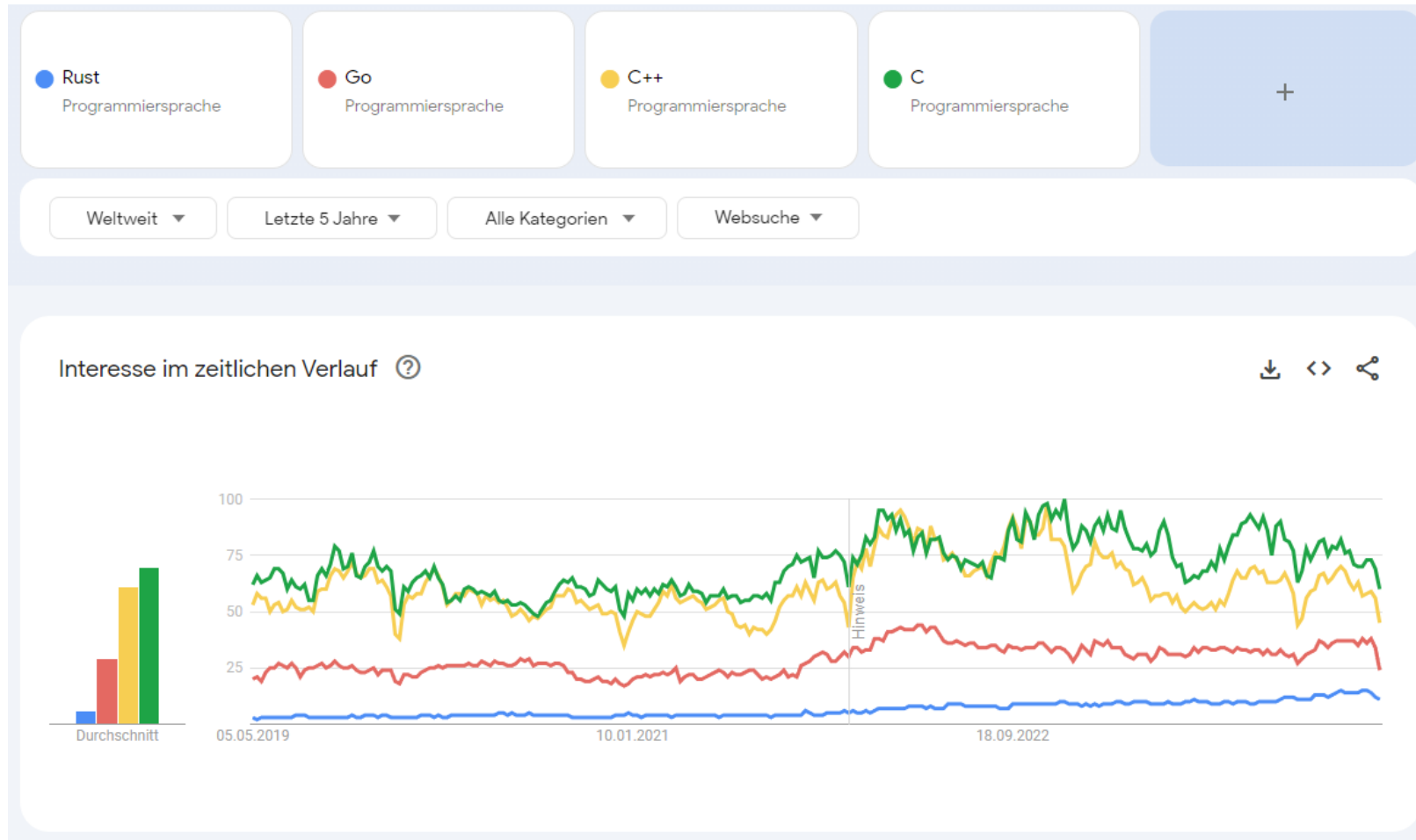


Associate



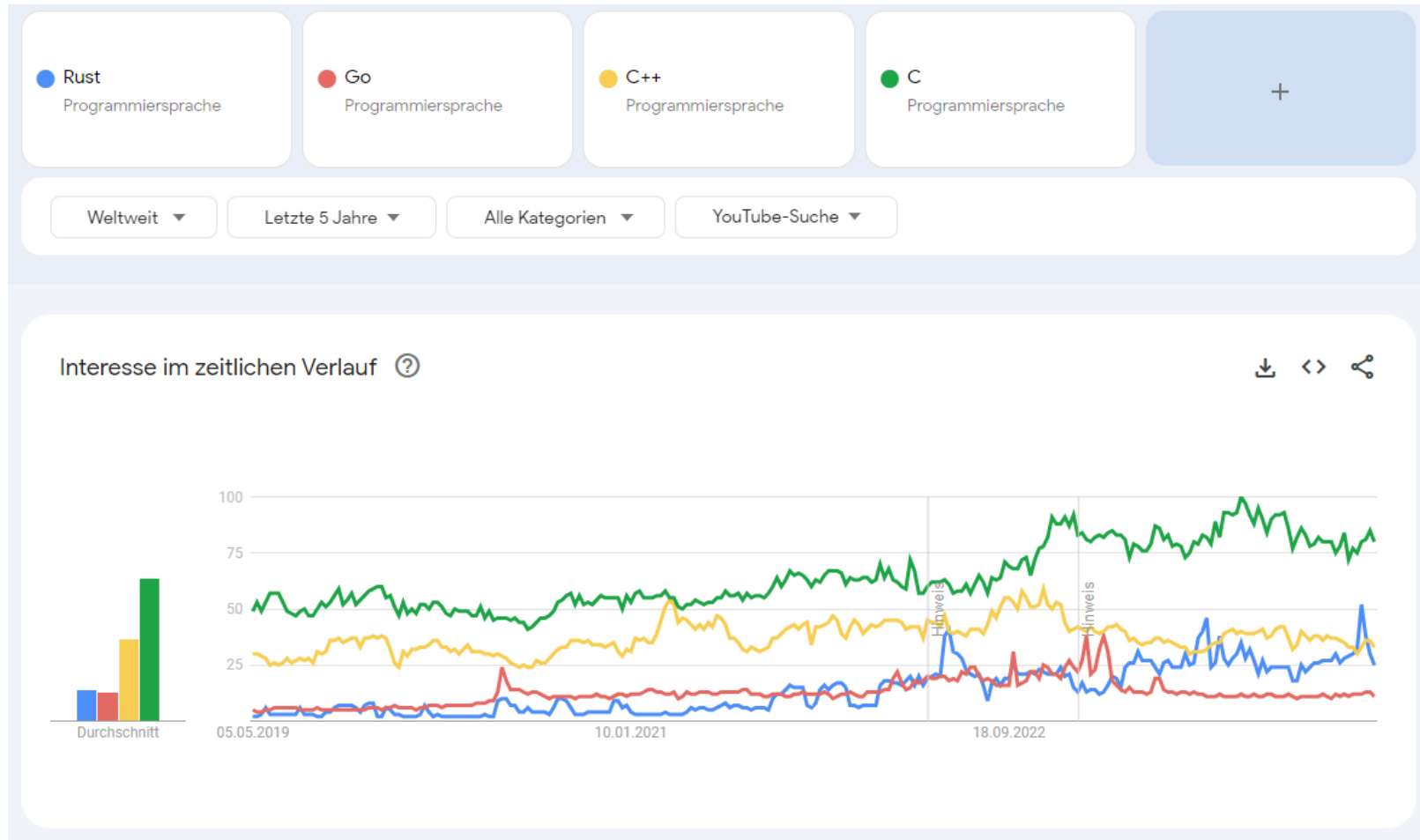
Quelle: <https://foundation.rust-lang.org/members/>

Ist Rust im Trend?



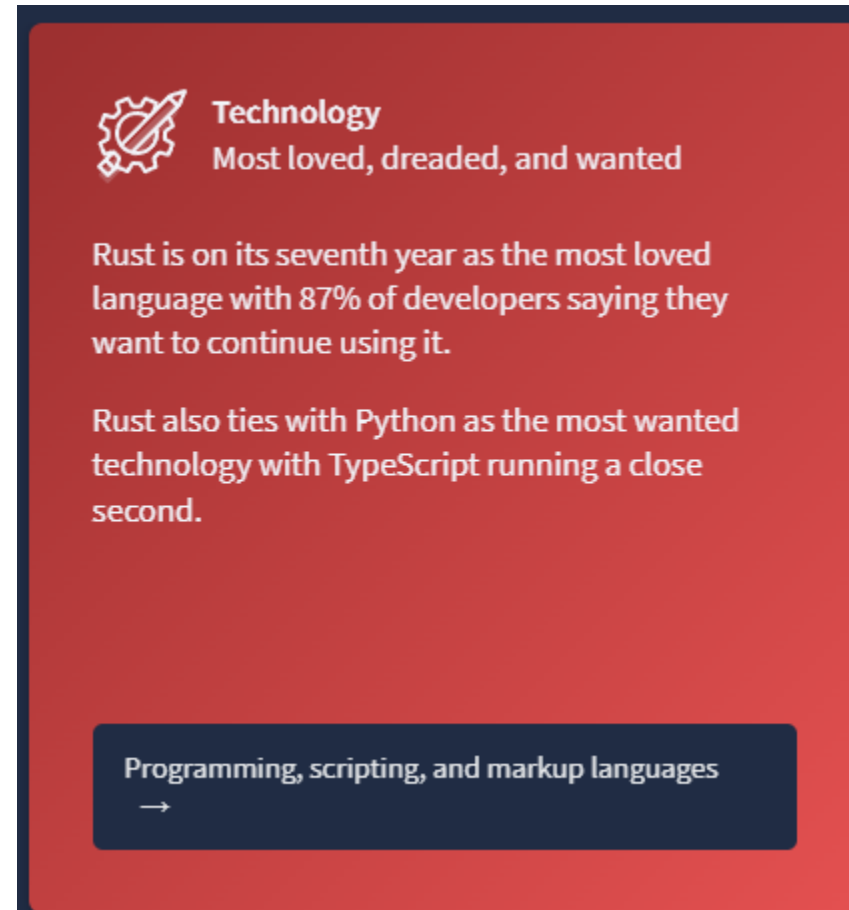
Quelle: <https://trends.google.com/>

Ist Rust im Trend?



Quelle: <https://trends.google.com/>

Ist Rust im Trend?



Quelle: <https://survey.stackoverflow.co/2022/#overview>

Community und Ökosystem

- Aktive und wachsende Community
- Zahlreiche Bibliotheken (145.329¹), sogenannte „Crates“

1. Stand vom 08.05.2024

13.05.2024

Andre-Johannes Müller

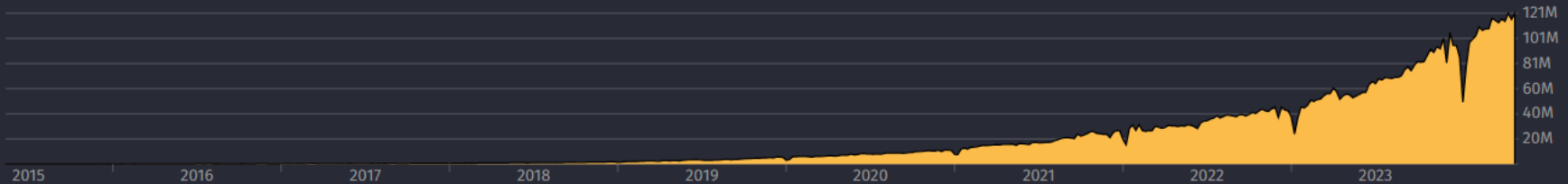
Web Technologies – Prof. Dr. Christian Noss

Page 9

TH Köln – Fakultät 10

Community und Ökosystem

Growth of the crates.io registry

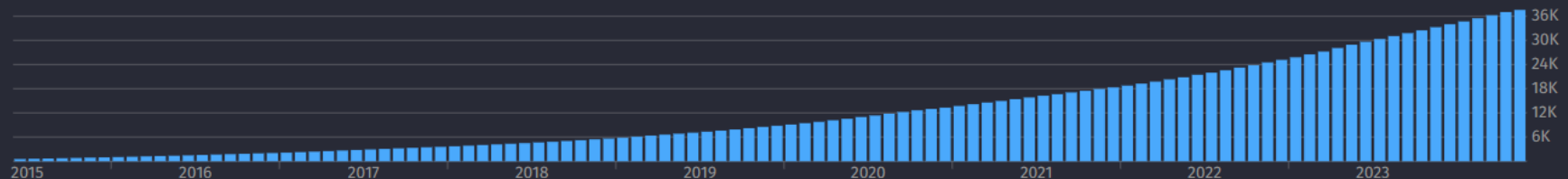


Daily downloads since Rust 1.0, 7-day average

Crate downloads are growing at a rate of 2.0× per year.

crates.io has served 149.5 million downloads in a *single day*, which is more than all downloads in the first *25 months* since the release of Rust 1.0 in May 2015.

Traffic during weekdays is typically 2.8× higher than during weekends (up from 2.6× a year before).



Number of users/teams owning a crate on crates.io

Quelle: <https://lib.rs/stats>

Wie ist die Performance? Rust vs Go

fannkuch-redux

source	secs	mem	gz	cpu secs
<u>Rust #6</u>	4.00	19,652	1260	15.86
<u>Rust #4</u>	7.10	19,652	1026	27.94
<u>Rust #5</u>	7.88	19,784	1023	30.85
<u>Go #3</u>	8.33	19,664	975	33.25
<u>Go</u>	11.74	19,664	906	46.77
<u>Go #2</u>	11.79	19,664	903	46.97
<u>Rust #2</u>	18.53	19,780	1198	73.32

n-body

source	secs	mem	gz	cpu secs
<u>Rust #9</u>	2.20	19,660	1881	2.20
<u>Rust #7</u>	3.13	19,660	1759	3.13
<u>Rust #2</u>	3.92	19,660	1809	3.92
<u>Rust #6</u>	3.95	19,788	1796	3.95
<u>Rust #8</u>	4.51	19,788	1774	4.50
<u>Rust #5</u>	5.06	19,788	2224	5.06
<u>Rust #4</u>	5.21	19,660	1810	5.21
<u>Rust</u>	5.51	19,660	1483	5.51
<u>Rust #3</u>	5.60	19,656	1546	5.60
<u>Go #3</u>	6.37	19,808	1207	6.38
<u>Go</u>	7.00	19,680	1316	7.02
<u>Go #2</u>	7.02	19,680	1222	7.03

mandelbrot

source	secs	mem	gz	cpu secs
<u>Rust #4</u>	0.96	33,280	1301	3.78
<u>Rust #6</u>	1.07	34,048	1338	4.20
<u>Rust #8</u>	1.06	33,280	770	4.21
<u>Rust #7</u>	1.06	33,280	763	4.21
<u>Rust #5</u>	1.12	33,920	725	4.40
<u>Rust #3</u>	1.14	19,656	1013	4.51
<u>Rust</u>	1.83	39,296	874	7.20
<u>Go #4</u>	3.76	35,328	912	14.92
<u>Go #3</u>	3.76	35,456	900	14.94
<u>Go</u>	5.01	33,408	829	19.98
<u>Go #2</u>	6.85	33,152	843	27.29
<u>Go #6</u>	6.89	32,768	707	27.40

gz: Anzahl von Bytes der komprimierten GZip Quellcode Datei

Quelle: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-go.html>

Wie ist die Performance? Rust vs C

fannkuch-redux					n-body					mandelbrot				
source	secs	mem	gz	cpu secs	source	secs	mem	gz	cpu secs	source	secs	mem	gz	cpu secs
C gcc #6	2.16	19,516	1582	8.53	C gcc #9	2.10	19,520	1639	2.10	Rust #4	0.96	33,280	1301	3.78
C gcc #4	14.20	19,516	1190	14.20	Rust #9	2.20	19,660	1881	2.20	Rust #6	1.07	34,048	1338	4.20
Rust #6	4.00	19,652	1260	15.86	Rust #7	3.13	19,660	1759	3.13	Rust #8	1.06	33,280	770	4.21
C gcc #5	6.91	19,644	917	27.13	C gcc #8	3.70	19,520	1398	3.69	Rust #7	1.06	33,280	763	4.21
Rust #4	7.10	19,652	1026	27.94	Rust #2	3.92	19,660	1809	3.92	Rust #5	1.12	33,920	725	4.40
Rust #5	7.88	19,784	1023	30.85	Rust #6	3.95	19,788	1796	3.95	Rust #3	1.14	19,656	1013	4.51
C gcc #3	39.57	19,516	574	39.57	C gcc #4	4.30	19,520	1496	4.30	C gcc #6	1.28	33,280	1141	5.11
C gcc	42.24	19,640	514	42.24	Rust #8	4.51	19,788	1774	4.50	C gcc #8	1.63	33,536	788	6.30
C gcc #2	11.16	19,644	1563	43.66	C gcc #6	4.96	19,520	1186	4.96	Rust	1.83	39,296	874	7.20
Rust #2	18.53	19,780	1198	73.32	Rust #5	5.06	19,788	2224	5.06	C gcc #4	3.48	33,152	805	13.85
					C gcc	5.18	19,520	1179	5.18	C gcc	3.49	32,896	828	13.88
					C gcc #2	5.18	19,520	1270	5.18	C gcc #7	3.49	33,152	1000	13.89
					Rust #4	5.21	19,660	1810	5.21	C gcc #3	3.54	32,768	769	14.07
					C gcc #3	5.24	19,520	1214	5.24	C gcc #9	4.22	33,280	700	16.30
					C gcc #5	5.41	19,648	1436	5.41	C gcc #2	24.97	19,520	412	24.97
					Rust	5.51	19,660	1483	5.51					
					Rust #3	5.60	19,656	1546	5.60					
					C gcc #7	6.81	19,520	1250	6.81					

gz: Anzahl von Bytes der komprimierten GZip Quellcode Datei

Quelle: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-go.html>

Data Races und wie Rust sie verhindert

- Data Races treten auf, wenn zwei oder mehr Threads gleichzeitig auf dieselbe Speicherstelle zugreifen und mindestens einer der Zugriffe schreibend ist, ohne dass eine geeignete Synchronisation stattfindet. Dies kann zu unvorhersehbaren Fehlern und instabilem Verhalten führen.
- Rusts Ansatz zur Vermeidung von Data Races:
 - Ownership-System: Jedes Datenstück in Rust hat einen eindeutigen Eigentümer. Die Ownership kann zwischen Funktionen oder Threads übertragen, aber nicht geteilt werden, es sei denn, es wird explizit zugelassen.
 - Borrow Checker: Überwacht Referenzen auf Daten und stellt sicher, dass entweder nur ein mutabler Zugriff oder mehrere unveränderliche Zugriffe gleichzeitig erfolgen.
 - Lifetime Prinzip: Jede Referenz in Rust hat eine Lebensdauer, die zur Compile-Zeit überprüft wird. Dies verhindert „Dangling“ Pointers und garantiert, dass Referenzen immer gültig sind.
- Vorteile:
 - Durch die strikte Einhaltung dieser Regeln stellt Rust sicher, dass Programme zur Laufzeit frei von Data Races sind. Dies führt zu zuverlässigeren und sichereren Anwendungen, besonders in Multithreading-Umgebungen.

Einführung in die Grundlagen

Rust Toolchain

Rust bringt verschiedene Werkzeuge (Tools) mit sich, die wichtigsten davon:

- **Rustc(Rust-Compiler):** rustc ist der Rust-Compiler, der für die Übersetzung von Rust-Quellcode in ausführbaren Maschinencode verantwortlich ist.
- **Cargo:** Cargo ist der Paketmanager und das „Build-System“ von Rust. Es vereinfacht den Prozess der Verwaltung von Abhängigkeiten, den Aufbau von Projekten und das Ausführen von Tests.
- **rustup:** rustup ist der Installer der Rust-Toolchain. Es ermöglicht einfach zwischen verschiedenen Versionen von Rust zu wechseln und die Toolchain aktuell zu halten.

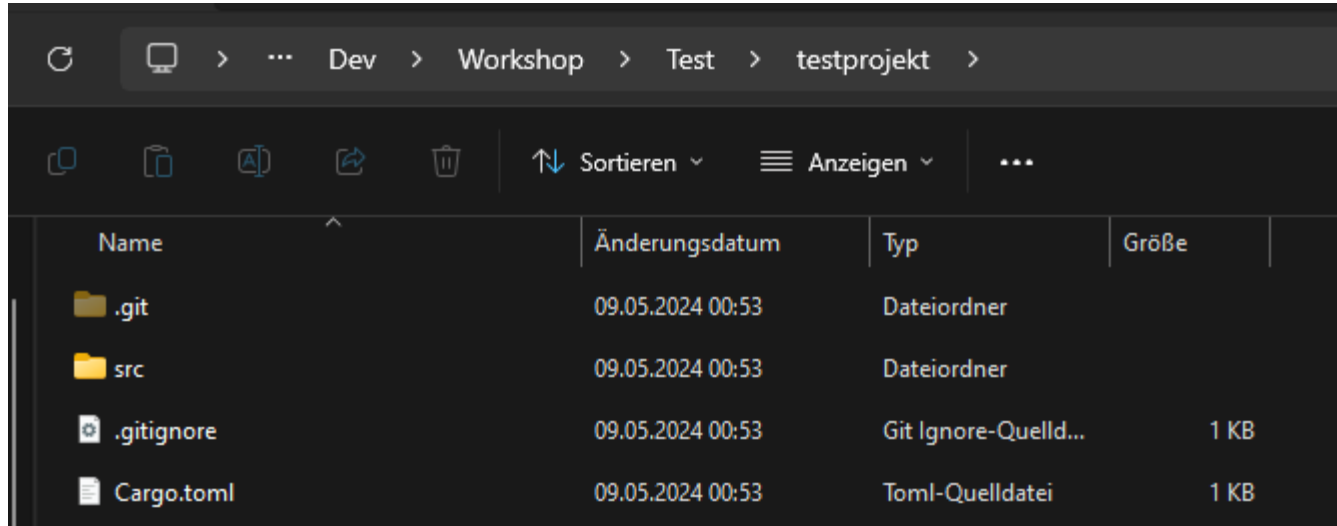
Paketmanager – Cargo

Cargo ist ein zentraler Bestandteil des Rust-Ökosystems und erfüllt mehrere Zwecke:

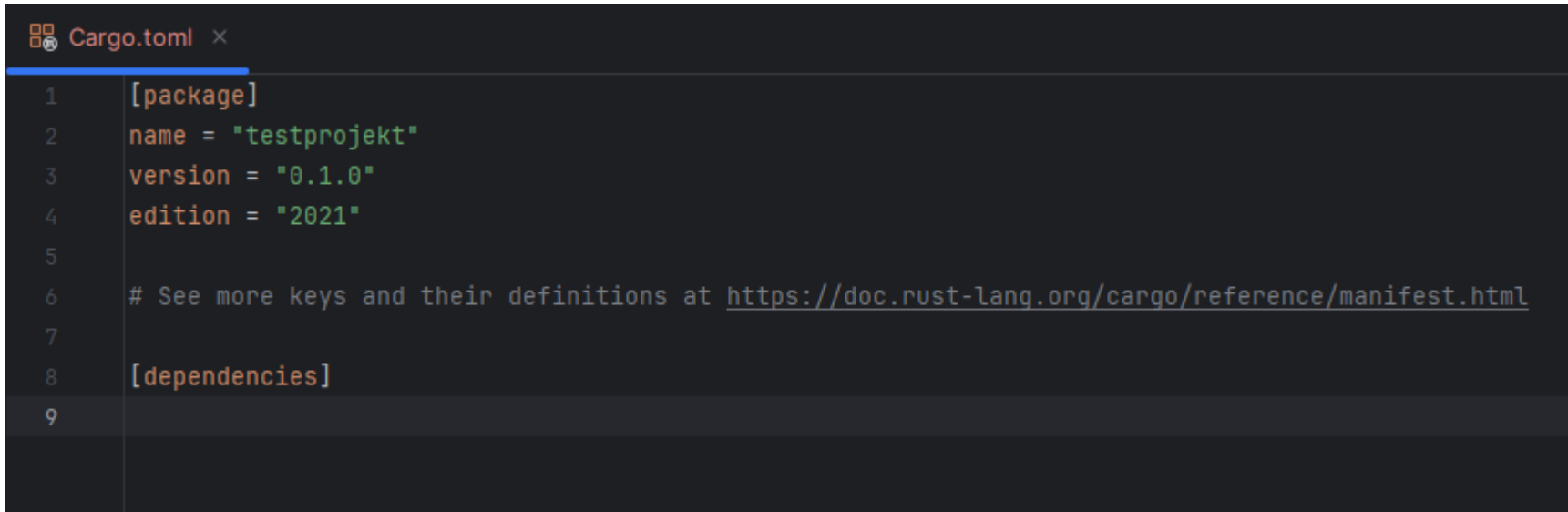
- **Abhängigkeitsmanagement:** Cargo verwaltet die Abhängigkeiten eines Projekts durch Angaben in der `Cargo.toml` Datei. Es holt und baut automatisch die benötigten Abhängigkeiten für Ihr Projekt.
- **Projektbau:** Cargo vereinfacht den Prozess des Kompilierens von Rust-Projekten. Benutzung von `cargo build (--release)` zum Kompilieren und `cargo run` zum Ausführen des Projekts.
- **Testen:** Cargo unterstützt das Schreiben und Ausführen von Tests integriert. Mit dem Befehl `cargo test` können Tests im Projekt ausgeführt werden.
- **Dokumentation:** Cargo integriert sich in das Dokumentationssystem von Rust. Mit `cargo doc` kann eine Dokumentation für das Projekt und dessen Abhängigkeiten erstellt und angezeigt werden.
- **Projekterstellung:** Mit `cargo new <Projektname>` lässt sich leicht ein neues Projekt starten. Dieser Befehl richtet ein neues Rust-Projekt mit einer grundlegenden Verzeichnisstruktur und Standardkonfiguration ein.

Paketmanager – Cargo new testprojekt

```
Andre > Test 0ms cargo new testprojekt
Created binary (application) `testprojekt` package
Andre > Test 572ms |
```

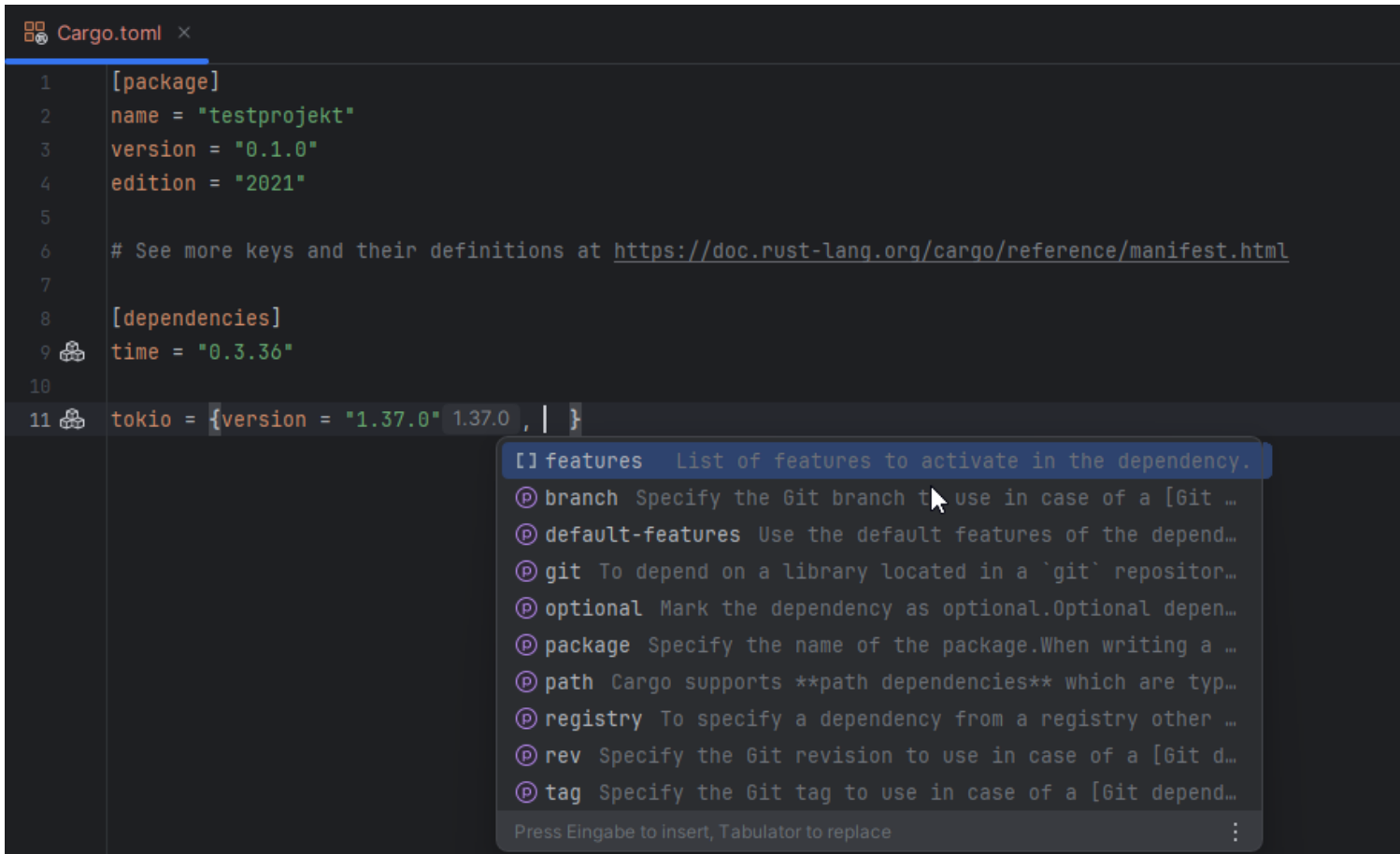


Paketmanager – Cargo.toml

A screenshot of a code editor showing a Cargo.toml file. The editor has a dark theme and a tab labeled 'Cargo.toml' with a close button. The code is as follows:

```
1 [package]
2   name = "testprojekt"
3   version = "0.1.0"
4   edition = "2021"
5
6   # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
7
8   [dependencies]
9
```

Paketmanager – Cargo.toml



```
1 [package]
2   name = "testprojekt"
3   version = "0.1.0"
4   edition = "2021"
5
6   # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
7
8 [dependencies]
9 time = "0.3.36"
10
11 tokio = {version = "1.37.0", features = ["full"], }
```

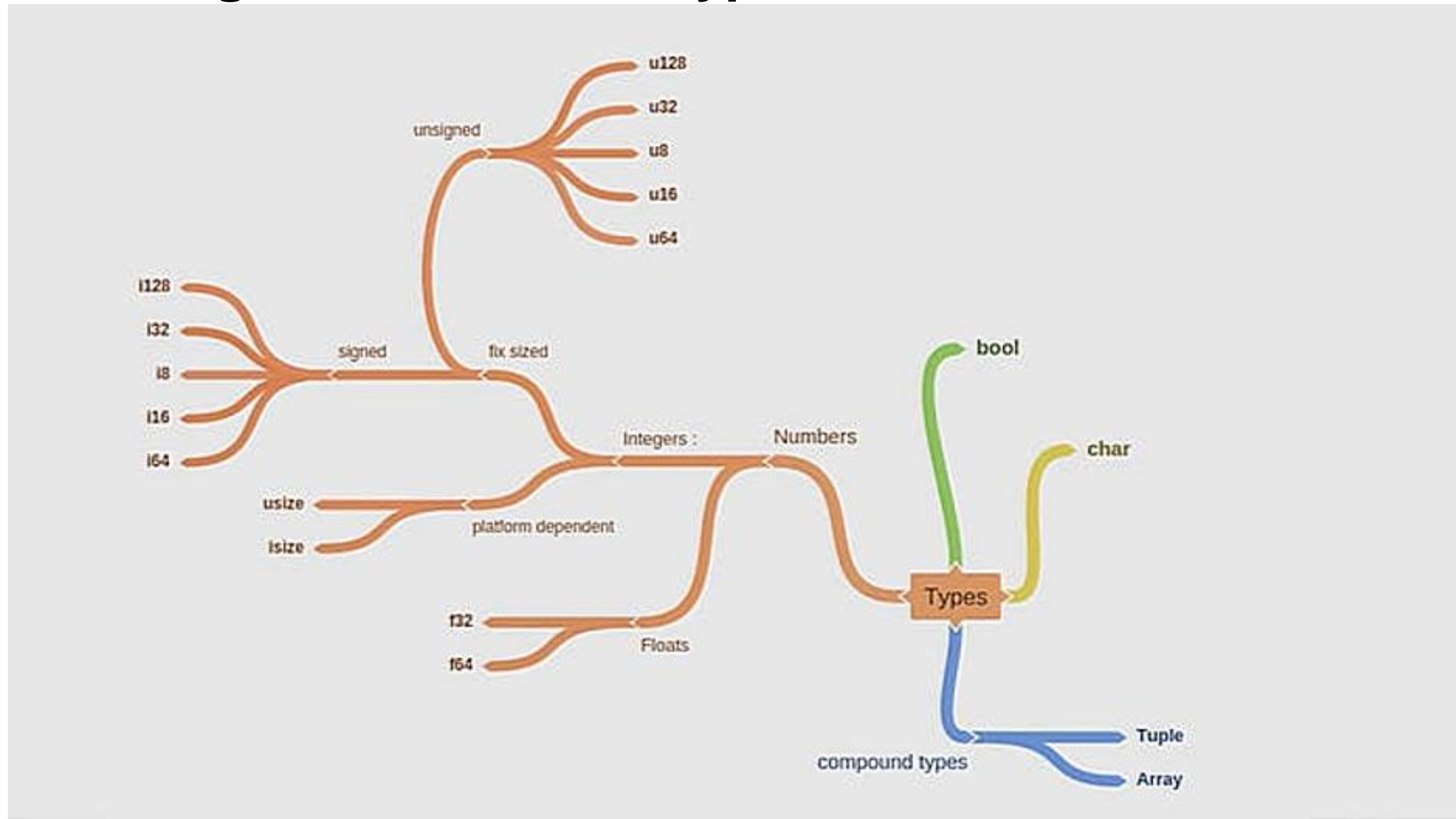
Tooltip for 'features' field:

- [] features List of features to activate in the dependency.
- Ⓟ branch Specify the Git branch to use in case of a [Git ...
- Ⓟ default-features Use the default features of the depend...
- Ⓟ git To depend on a library located in a 'git' repositor...
- Ⓟ optional Mark the dependency as optional.Optional depen...
- Ⓟ package Specify the name of the package.When writing a ...
- Ⓟ path Cargo supports **path dependencies** which are typ...
- Ⓟ registry To specify a dependency from a registry other ...
- Ⓟ rev Specify the Git revision to use in case of a [Git d...
- Ⓟ tag Specify the Git tag to use in case of a [Git depend...

Press Eingabe to insert, Tabulator to replace

Öffnen von Vorschlägen mit:
STRG + Leertaste

Grundlagen – Basic Datentypen



Grundlagen – Aufbau „Hello World“-Programm



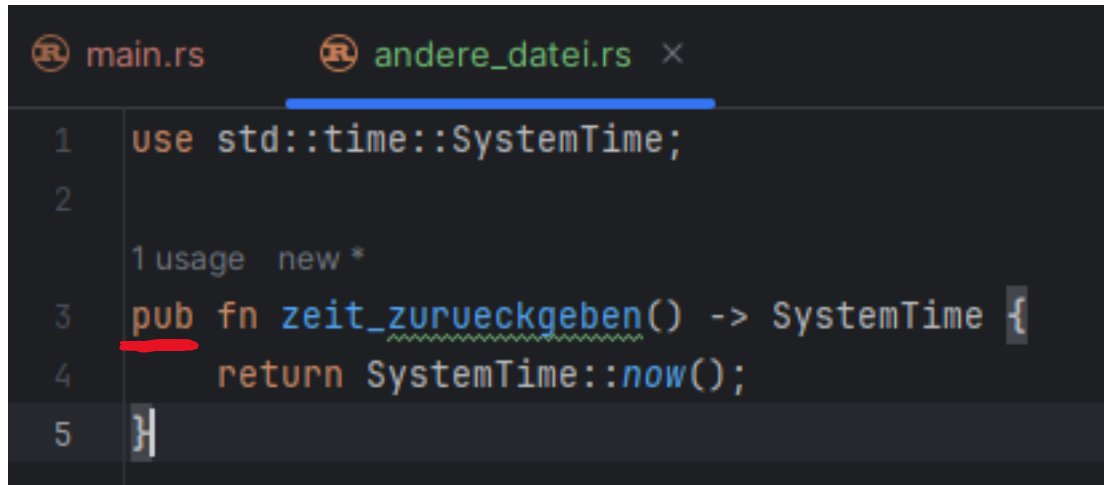
```
1 fn main() {
2     println!("Hello World");
3     another_function();
4 }
5
6 fn another_function() {
7     println!("Another function.");
8 }
```

- `main.rs` oder `lib.rs` muss unter **src/** vorhanden sein
- Die `main` Funktion ist der erste Code der in einer Rust Executable läuft
- Alles mit **funktion!** ist ein **Macro**
 - **Macros** sind ein Weg Code zu schreiben, der anderen Code schreibt, sogenanntes „Meta Programming“

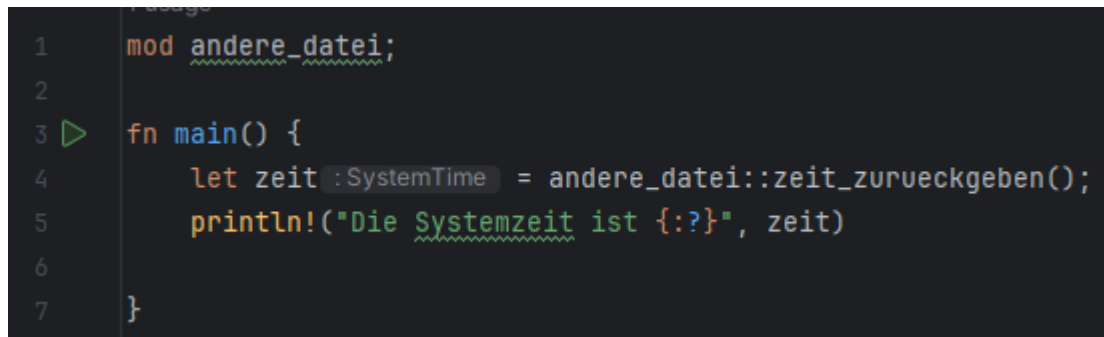
Grundlagen – Importieren von Crates / Funktionen

```
1  /*
2  Kommentar der etwas erklären könnte.
3  */
4  use std::time::SystemTime;
5
6  /*
7  Code der in andere Dateien ausgelagert wurde muss über das "mod" keyword in den Scope geholt werden
8  */
9
10 // mod andere_datei;
11 ▶ fn main() {
12     let zeit : SystemTime = SystemTime::now();
13     println!("Die Systemzeit ist {:?}", zeit)
14
15 }
```

Grundlagen – Importieren von Crates / Funktionen



```
main.rs  andere_datei.rs x
1 use std::time::SystemTime;
2
3 pub fn zeit_zurueckgeben() -> SystemTime {
4     return SystemTime::now();
5 }
```



```
1 mod andere_datei;
2
3 fn main() {
4     let zeit : SystemTime = andere_datei::zeit_zurueckgeben();
5     println!("Die Systemzeit ist {:?}", zeit)
6
7 }
```

Grundlagen – Variablen

```
fn main() {  
    let x = 5;  
    println!("The value of x is: {x}");  
    x = 6;  
    println!("The value of x is: {x}");  
}
```



Cargo run liefert:

```
Compiling testprojekt v0.1.0 (C:\Dev\Workshop\Test\testprojekt)  
error[E0384]: cannot assign twice to immutable variable `x`  
--> src\main.rs:4:5  
  |  
2 |     let x = 5;  
  |     -  
  |     |  
  |     first assignment to `x`  
  |     help: consider making this binding mutable: `mut x`  
3 |     println!("The value of x is : {x}");  
4 |     x = 6;  
  |     ^^^^^ cannot assign twice to immutable variable  
  
For more information about this error, try `rustc --explain E0384`.  
error: could not compile `testprojekt` (bin "testprojekt") due to 1 previous error
```


Grundlagen – Variablen – Mutable Keyword

```
fn main() {  
    let mut x = 5;  
    println!("The value of x is: {x}");  
    x = 6;  
    println!("The value of x is: {x}");  
}
```

```
$ cargo run  
  Compiling variables v0.1.0 (file:///projects/variables)  
  Finished dev [unoptimized + debuginfo] target(s) in 0.30s  
  Running `target/debug/variables`  
The value of x is: 5  
The value of x is: 6
```

Grundlagen – Grundaufbau Funktion

```
fn funktions_name(param1 : Typ, param2: Typ) -> ReturnType {  
    ReturnType{ }  
}
```

```
fn funktion_ohne_return_type(param1: Typ, param2: Typ) -> () {  
  
}  
  
fn auch_funktion_ohne_return_type_implicit(param1: Typ) {  
  
}
```

Grundlagen – If Anweisungen

```
fn main() {  
    let number = 3;  
  
    if number < 5 {  
        println!("condition was true");  
    } else {  
        println!("condition was false");  
    }  
}
```

- If Bedingungen müssen vom Typ bool sein, folgendes **geht nicht**

```
fn main() {  
    let number = 3;  
  
    if number {  
        println!("number was three");  
    }  
}
```

Grundlagen – Else If Abfragen

```
fn main() {  
    let number = 6;  
  
    if number % 4 == 0 {  
        println!("number is divisible by 4");  
    } else if number % 3 == 0 {  
        println!("number is divisible by 3");  
    } else if number % 2 == 0 {  
        println!("number is divisible by 2");  
    } else {  
        println!("number is not divisible by 4, 3, or 2");  
    }  
}
```

Grundlagen – Variablen Zuweisung mit If

```
fn main() {  
    let condition = true;  
    let number = if condition { 5 } else { 6 };  
}
```

Grundlagen – Schleifen

- Es gibt 3 Schleifenarten
 - **loop** (wie while true)
 - **while**
 - **for**

Grundlagen – Iterieren - Loop

```
fn main() {  
    loop {  
        println!("again!");  
    }  
}
```

```
$ cargo run  
  Compiling loops v0.1.0 (file:///projects/loops)  
  Finished dev [unoptimized + debuginfo] target(s) in 0.29s  
  Running `target/debug/loops`  
again!  
again!  
again!  
again!  
^Cagain!
```

Grundlagen – Iterieren - Variablenzuweisung durch loop

```
fn main() {  
    let mut counter = 0;  
  
    let result = loop {  
        counter += 1;  
  
        if counter == 10 {  
            break counter * 2;  
        }  
    };  
  
    println!("The result is {result}");  
}
```

The result is 20

Grundlagen – Iterieren - While

```
fn main() {  
    let a = [10, 20, 30, 40, 50];  
    let mut index = 0;  
  
    while index < 5 {  
        println!("the value is: {}", a[index]);  
  
        index += 1;  
    }  
}
```

Grundlagen – Iterieren – For (each)

```
fn main() {  
    let a = [10, 20, 30, 40, 50];  
  
    for element in a {  
        println!("the value is: {element}");  
    }  
}
```

Grundlagen – Iterieren – Range For Loop

```
fn main() {  
    for number in (1..4).rev() {  
        println!("{number}!");  
    }  
    println!("LIFTOFF!!!");  
}
```

```
3!  
2!  
1!  
LIFTOFF!!!
```

```
fn main() {  
    for number in (1..=4).rev() {  
        println!("{number}!");  
    }  
    println!("LIFTOFF!!!");  
}
```

```
4!  
3!  
2!  
1!  
LIFTOFF!!!
```

Grundlagen – Collections & Strings

```
9 ▶ fn main() {
10     let arr_zahlen : [i32; 3] = [1, 2, 3];
11     println!("{:?}", arr_zahlen);
12
13     let arr_str : [&str; 2] = ["Hallo", "Welt"];
14     // Einem Array kann nach der Zuweisung nichts neues hinzugefügt werden!
15     println!("{:?}", arr_str);
16
17     let string_1 : String = String::from(s: "Hallo");
18     let mut string_2 : String = String::new();
19     string_2.push_str(string: " , Welt");
20
21     let arr_strings : [String; 2] = [string_1, string_2];
22     println!("{:?}", arr_strings);
23
24     let mut vector : Vec<i32> = vec!(1, 2, 3); // Nichts anderes als ein Dynamisches Array
25     vector.push(value: 4);
26     println!("{:?}", vector)
27 }
28
```

Grundlagen – Ausdrücke

```
9 ▶ fn main() {  
10     let x:u32 = 5u32; // ist das selbe wie let x: u32 = 5;  
11  
12     let y:u32 = {  
13         let x_quadriert:u32 = x * x;  
14         let x_würfel_volumen:u32 = x_quadriert * x; // das selbe wie x^3  
15  
16         // Dieser Ausdruck wird y zugewiesen  
17         x_würfel_volumen + x_quadriert + x  
18     };  
19  
20     let z:() = {  
21         // Das Semikolon unterdrückt diesen Ausdruck und '()' wird zugewiesen zu 'z'  
22         2 * x;  
23     };  
24  
25     println!("x ist {:?}", x);  
26     println!("y ist {:?}", y);  
27     println!("z ist {:?}", z);  
28 }
```

```
x ist 5  
y ist 155  
z ist ()
```

Grundlagen – Ausdrücke

```
1  ▶ fn main() {  
2      let number : i32 = 42;  
3  
4      let category : &str = match number {  
5          0 => "Null",  
6          1..=10 => "Kleiner als 10",  
7          11..=50 => "Zwischen 11 und 50",  
8          _ => "Größer als 50",  
9      };  
10  
11      println!("Die Kategorie von {} ist: {}", number, category);  
12  }
```

Grundlagen – Result Enum

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

- Funktionen können ein Result zurückgeben, um den Erfolg oder Fehler der Ausführung zu propagieren.
- Wird häufig bei I/O Operationen verwendet.

Grundlagen – Result Handling mit match

```
1  fn divide(x: i32, y: i32) -> Result<i32, &'static str> {
2      if y == 0 {
3          Err("Division durch Null ist nicht erlaubt.")
4      } else {
5          Ok(x / y)
6      }
7  }
8
9  fn main() {
10     let result : Result<i32, &str> = divide( x: 10, y: 2);
11
12     match result {
13         Ok(value : i32) => println!("Ergebnis: {}", value),
14         Err(msg : &str) => println!("Fehler: {}", msg),
15     }
16 }
```


Grundlagen – Result Handling mit if let

```
1  fn divide(x: i32, y: i32) -> Result<i32, &'static str> {
2      if y == 0 {
3          Err("Division durch Null ist nicht erlaubt.")
4      } else {
5          Ok(x / y)
6      }
7  }
8
9  ▶ fn main() {
10     let result : Result<i32, &str> = divide( x: 10, y: 0);
11
12     if let Err(e : &str) = result {
13         panic!("Ein Fehler der so kritisch war, dass das Programm jetzt stoppen muss ist passiert! {}", e)
14     }
15
16     if let Ok(ergebnis_division : i32) = result {
17         println!("Das Ergebnis der Division ist {}", ergebnis_division)
18     }
19 }
```

Grundlagen – Panic Fall

Ein Fehler der so kritisch war, dass das Programm jetzt stoppen muss ist passiert! Division durch Null ist nicht erlaubt.
stack backtrace:

```
0: std::panicking::begin_panic_handler
   at /rustc/25ef9e3d85d934b27d9dada2f9dd52b1dc63bb04/library/std/src/panicking.rs:647
1: core::panicking::panic_fmt
   at /rustc/25ef9e3d85d934b27d9dada2f9dd52b1dc63bb04/library/core/src/panicking.rs:72
2: testprojekt::main
   at .\src\main.rs:13
3: core::ops::function::FnOnce::call_once<void (*)(),tuple$<> >
   at /rustc/25ef9e3d85d934b27d9dada2f9dd52b1dc63bb04/library/core/src/ops/function.rs:250
4: core::hint::black_box
```

Grundlagen – ? Operator

```
fn write_info(info: &Info) -> io::Result<()> {  
    // Early return on error  
    let mut file = match File::create("my_best_friends.txt") {  
        Err(e) => return Err(e),  
        Ok(f) => f,  
    };  
    if let Err(e) = file.write_all(format!("name: {}\n", info.name).as_bytes()) {  
        return Err(e)  
    }  
    if let Err(e) = file.write_all(format!("age: {}\n", info.age).as_bytes()) {  
        return Err(e)  
    }  
    if let Err(e) = file.write_all(format!("rating: {}\n", info.rating).as_bytes()) {  
        return Err(e)  
    }  
    Ok(())  
}
```

Grundlagen – ? Operator



```
fn write_info(info: &Info) -> io::Result<()> {  
    let mut file = File::create("my_best_friends.txt"?;  
    // Early return on error  
    file.write_all(format!("name: {}\n", info.name).as_bytes())?;  
    file.write_all(format!("age: {}\n", info.age).as_bytes())?;  
    file.write_all(format!("rating: {}\n", info.rating).as_bytes())?;  
    Ok()  
}
```

Grundlagen – Option Enum – Bessere Alternative zu Null

```
1 fn main() {  
2     // Optionales Ergebnis: Entweder eine Zahl oder keine  
3     let optional_number: Option<i32> = Some(42);  
4  
5     // Verwendung von match  
6     match optional_number {  
7         Some(number :i32) => println!("Die Zahl ist: {}", number),  
8         None => println!("Keine Zahl vorhanden"),  
9     }  
10  
11     // Verwendung von if let  
12     if let Some(number :i32) = optional_number {  
13         println!("Die Zahl ist: {}", number);  
14     } else {  
15         println!("Keine Zahl vorhanden");  
16     }  
17 }
```

Grundlagen – unwrap() – Options und Results

Wenn man sich 100% sicher ist, dass nur der „Happy Path“ eintritt, kann man auf Results und Options **.unwrap()** aufrufen, dann muss man das Error Handling nicht machen

```
1 use std::fs::File;
2
3 fn main() {
4     let file :File = File::open( path: "example.txt").unwrap();
5     println!("Datei geöffnet: {:?}", file);
6 }
```

```
3 fn main() {
4     let file_result :Result<File> = File::open( path: "example.txt");
5
6     match file_result {
7         Ok(file :File ) => {
8             // Datei erfolgreich geöffnet
9             println!("Datei geöffnet: {:?}", file);
10        }
11        Err(error :Error ) => {
12            // Fehler beim Öffnen der Datei
13            println!("Fehler beim Öffnen der Datei: {:?}", error);
14        }
15    }
```

Zeit für Aufgaben

Pause?

Structs

- Bieten Möglichkeit zum Gruppieren von zusammenhängenden Werten

```
struct User {  
    active: bool,  
    username: String,  
    email: String,  
    sign_in_count: u64,  
}
```

```
fn main() {  
    let user1 = User {  
        active: true,  
        username: String::from("someusername123"),  
        email: String::from("someone@example.com"),  
        sign_in_count: 1,  
    };  
}
```

Structs – Methoden hinzufügen via impl

```
#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}

fn main() {
    let rect1 = Rectangle {
        width: 30,
        height: 50,
    };

    println!(
        "The area of the rectangle is {} square pixels.",
        rect1.area()
    );
}
```

Das **#Derive(Debug)** Macro ermöglicht es Instanzen des Structs via **println!("{}", structName)** auszugeben

Structs – „Konstruktor“ Funktionen

- Meistens wird eine „new“-Methode angelegt, welche als Konstruktor des Structs dient.

```
impl Rectangle {  
    fn square(size: u32) -> Self {  
        Self {  
            width: size,  
            height: size,  
        }  
    }  
}
```

Memory Management Lösungen

	Pros	Kontras
Garbage Collection (Java, Javascript, Python ...)	<ul style="list-style-type: none"> • Fehlerfrei* • Schnellere Entwicklungszeit 	<ul style="list-style-type: none"> • Keine Kontrolle über den Speicher • Langsamere und unvorhersehbare Runtime Performance • Größere Programmgröße
Manuelles Memory Management (z.B. C, C++)	<ul style="list-style-type: none"> • Kontrolle über den Speicher • Schnellere Runtime Performance • Kleinere Programmgröße 	<ul style="list-style-type: none"> • Fehleranfällig • Langsamere Entwicklungszeit
Ownership Model (Rust)	<ul style="list-style-type: none"> • Kontrolle über den Speicher • Fehlerfrei* • Schnellere Runtime Performance • Kleinere Programmgröße 	<ul style="list-style-type: none"> • Langsamere Entwicklungszeit. Lernkurve steiler (Durch nötiges kämpfen mit dem Borrow Checker)

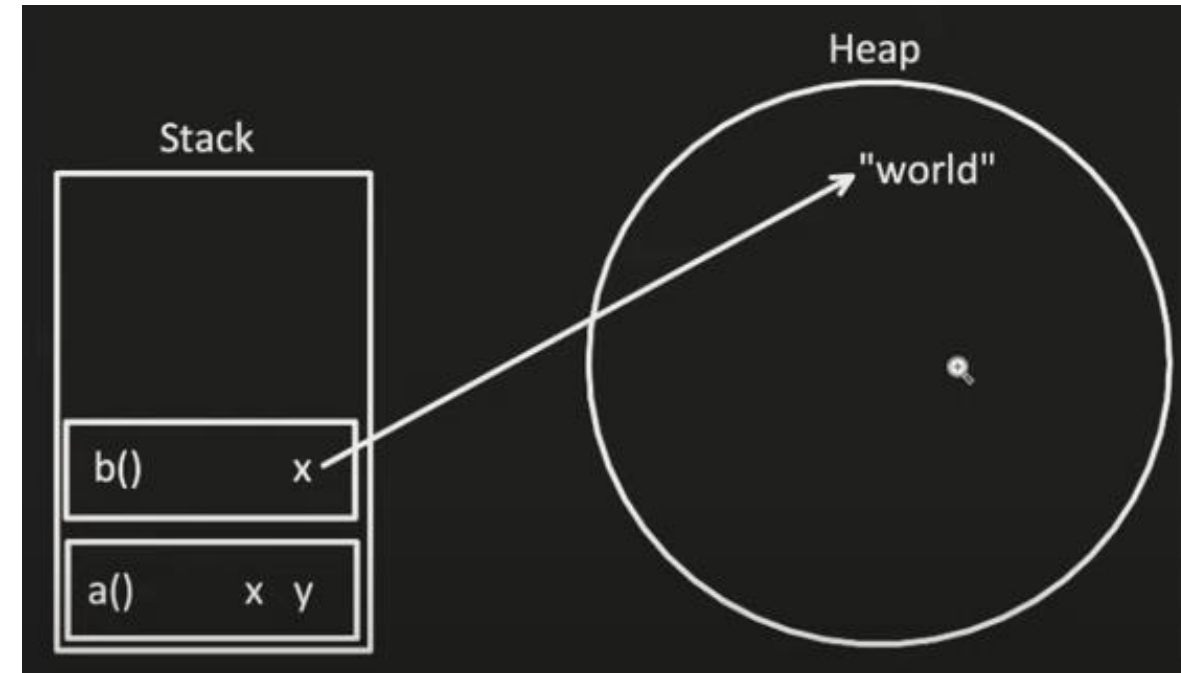
Stack & Heap - Grundverständnis

- Während der Laufzeit hat ein Programm Zugang zum Stack & Heap
- Der Stack hat eine feste Größe und kann während der Laufzeit nicht wachsen oder kleiner werden.
- Für jede Funktion während der Laufzeit wird ein sog. „Stack Frame“ angelegt (dazu gleich mehr)
- Während des Kompilierens wird die Größe eines Stack Frames berechnet, d.h. es muss während des kompilierens bereits die Größe der Variablen (fix!) im Stack bekannt sein
- Variablen auf dem Stack leben nur so lange wie der Stack Frame
- Heap ist unorganisiert und viel langsamer
- Heap kann während der Laufzeit wachsen oder kleiner werden

Weiterführende Quelle (für Zuhause): https://www.youtube.com/watch?v=_8-ht2AKyH4

Stack & Heap - Grundverständnis

```
1 fn main() {  
2     fn a() {  
3         let x: &str = "hello";  
4         let y: i32 = 22;  
5         b();  
6     }  
7  
8     fn b() {  
9         let x: String = String::from("world");  
10    }  
11 }
```



Weiterführende Quelle (für Zuhause): https://www.youtube.com/watch?v=_8-ht2AKyH4

Ownership Regeln

- Die Ownership Regeln ermöglichen Rust Memory Safety Garantien ohne einen Garbage Collector
- Die drei Regeln sind:
 - Jeder Wert in Rust hat eine Variable die dessen Besitzer (Owner) ist
 - Es kann nur einen Besitzer zur gleichen Zeit geben
 - Wenn der Owner out of scope geht, wird der Speicher des Wertes aufgeräumt (drop Funktion wird aufgerufen)

Ownership Regeln – Variablen Scope

```
3 ▶ fn main() {  
4  
5     { // s ist nicht valide hier, s wurde noch nicht angelegt  
6         let s : &str = "hello"; // s ist valide von diesem Punkt vorwärts  
7         // Mach irgendwas mit s...  
8     } // Dieser scope ist nun vorbei, s ist nicht mehr valide  
9 }
```

Auf dem Stack angelegt

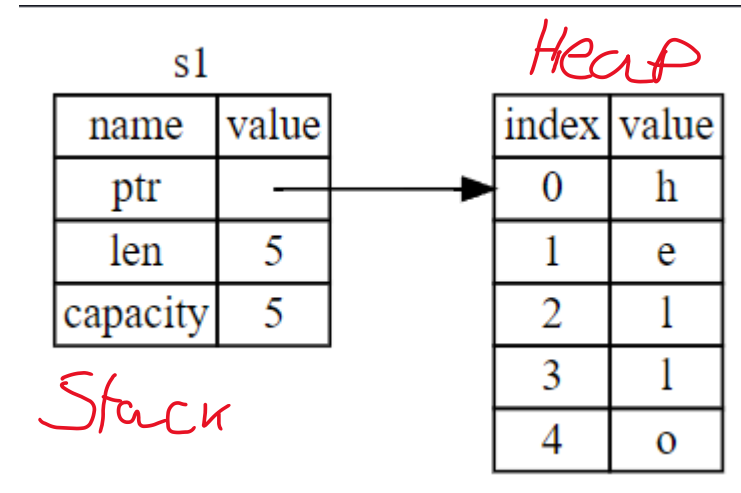
```
3 ▶ fn main() {  
4  
5     { // s ist nicht valide hier, s wurde noch nicht angelegt  
6         let s : String = String::from(s: "hello"); // s ist valide von diesem Punkt vorwärts  
7         // Mach irgendwas mit s...  
8     } // Dieser scope ist nun vorbei, s ist nicht mehr valide  
9 }
```

Auf dem Heap
angelegt.

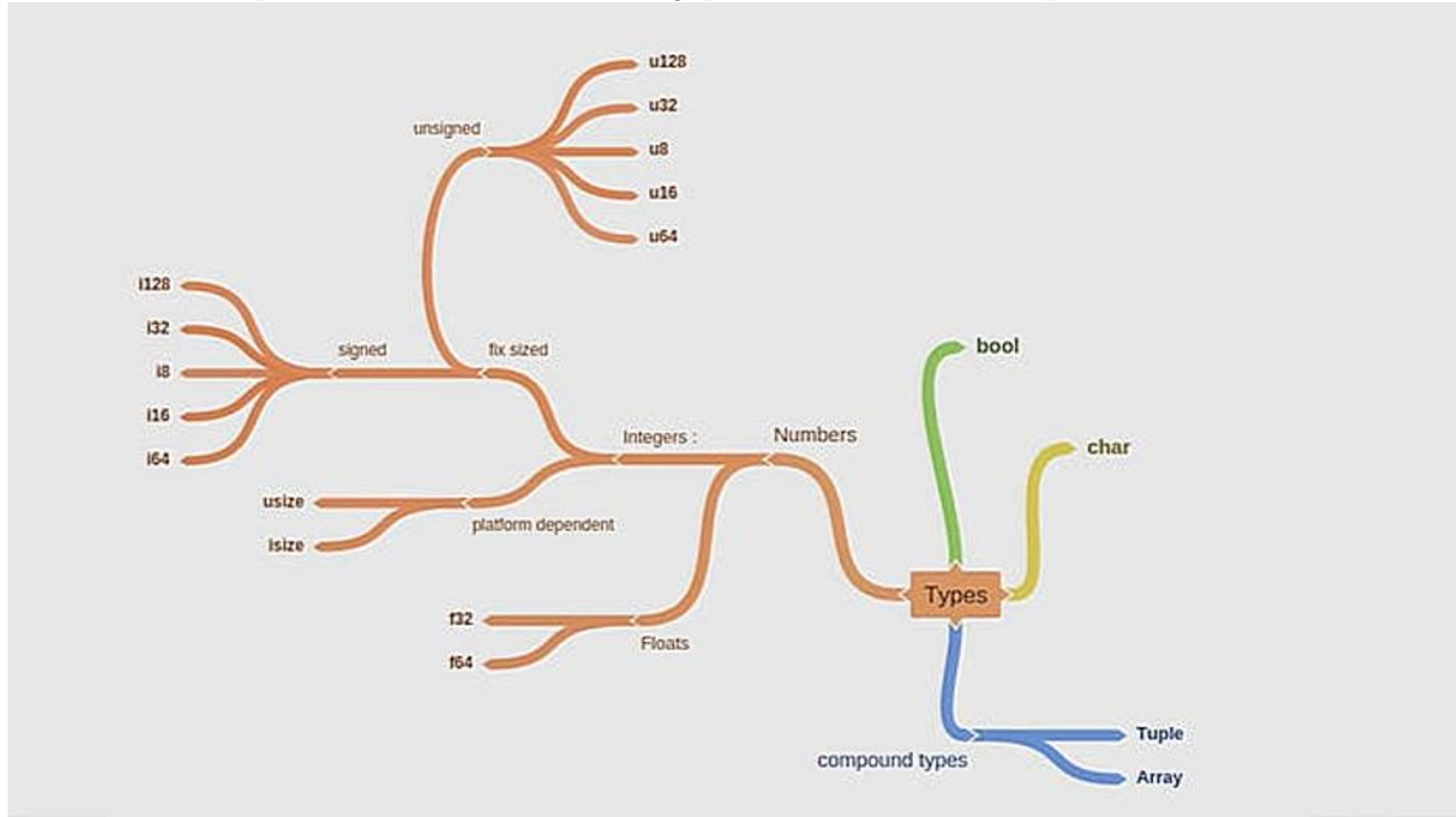
Andere Sprachen
benutzen
„new“ keyword

Ownership Regeln – Copy

```
3 ▶ fn main() {  
4     let x: i32 = 5;  
5     let y: i32 = x; //Copy  
6  
7     let s1: String = String::from(s: "hello");  
8     let s2: String = s1;  
9  
10 }
```



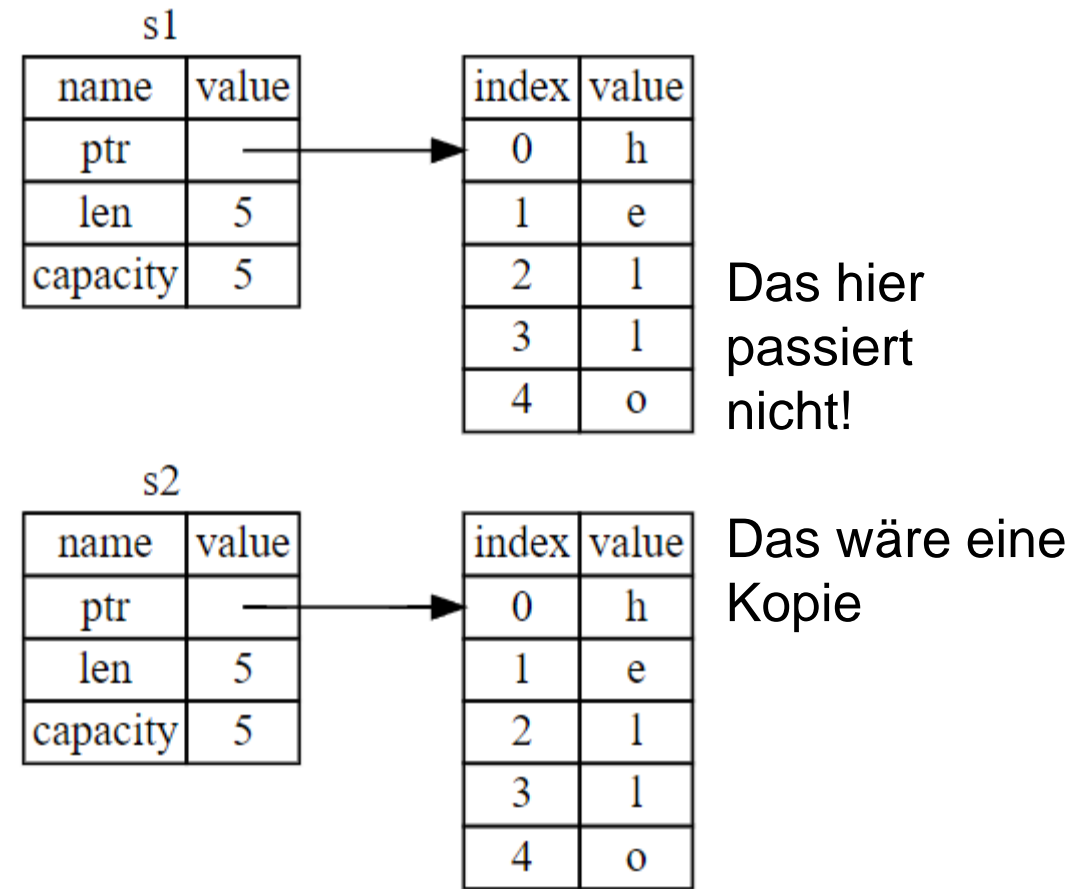
Ownership – Diese Datentypen + &str implementieren Copy



Ownership Regeln – Deep Copy

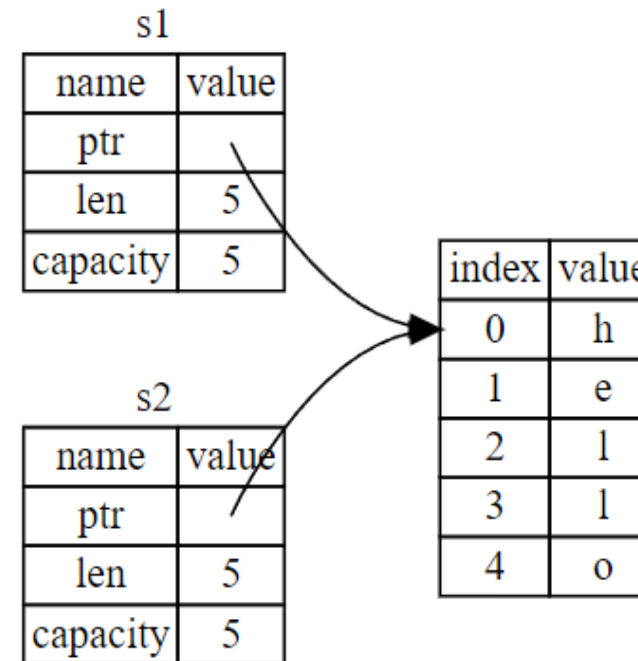
```

3  ▶ fn main() {
4      let x: i32 = 5;
5      let y: i32 = x; //Copy
6
7      let s1: String = String::from(s: "hello");
8      let s2: String = s1;
9
10 }
```



Ownership Regeln – Shallow - Copy

```
3 ▶ fn main() {  
4     let x: i32 = 5;  
5     let y: i32 = x; //Copy  
6  
7     let s1: String = String::from(s: "hello");  
8     let s2: String = s1;  
9  
10 }
```



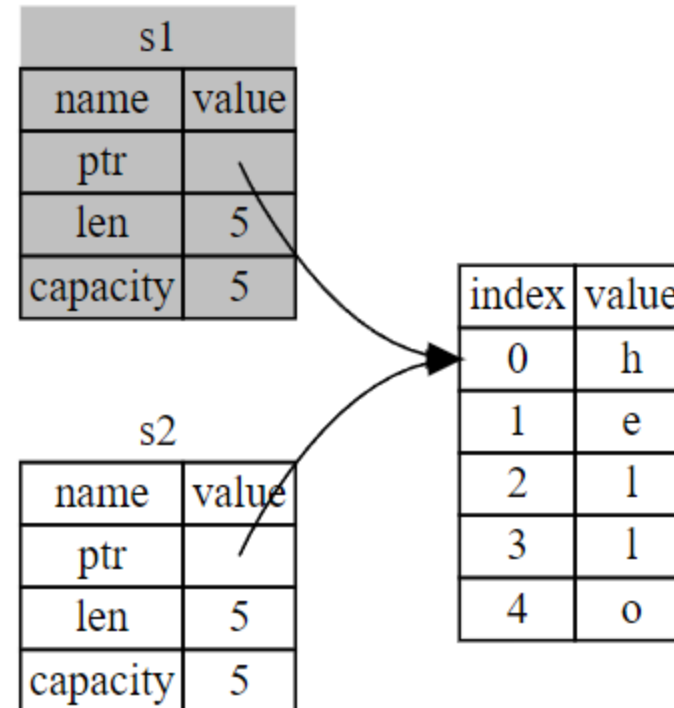
Das hier
passiert
nicht!

Das wäre eine
Shallow
Copy

Ownership Regeln – Move

```
3  main() {  
4      let x: i32 = 5;  
5      let y: i32 = x; //Copy  
6  
7      let s1: String = String::from(s: "hello");  
8      let s2: String = s1; //Move (keine Shallow Copy)  
9  
10     println!("{}", world!, s1);  
11  
12 }
```

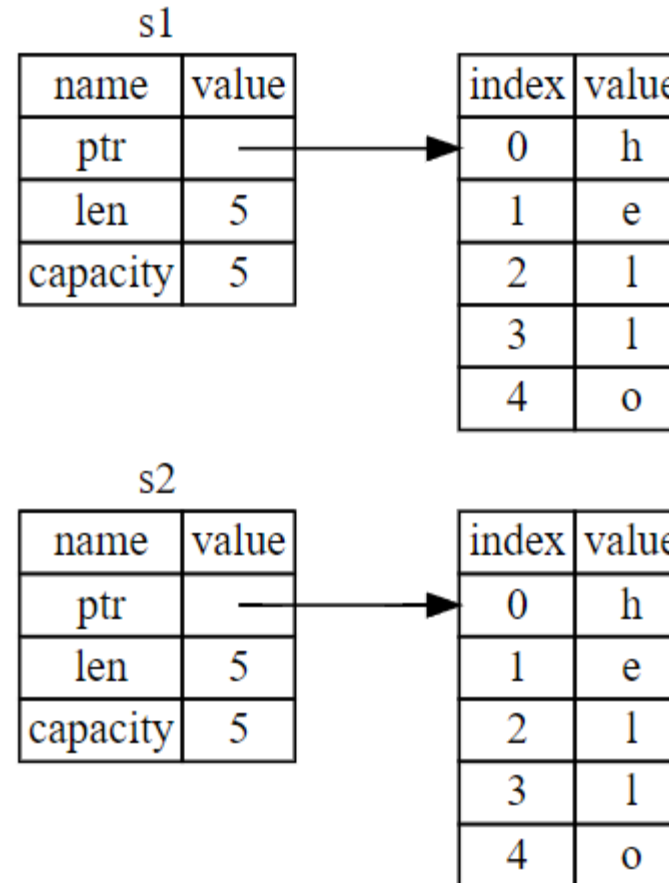
s1 kann nicht mehr benutzt werden!



Das passiert
in Rust automatisch

Ownership Regeln – Deep Copy

```
3 ▶ fn main() {  
4     let x: i32 = 5;  
5     let y: i32 = x; //Copy  
6  
7     let s1: String = String::from(s: "hello");  
8     let s2: String = s1.clone(); //Deep Copy  
9     println!("{}", world!, s1);  
10    println!("{}", world!, s2);  
11  
12 }
```



Ownership Regeln – Übergeben von Ownership

```
3 ▶ fn main() {  
4     let s1 :String = String::from(s: "hello");  
5     nimmt_ownership(s1);  
6     println!("{}", world!, s1);  
7  
8 }  
9  
1 usage  
10 fn nimmt_ownership(some_string : String){ //some_string wird gemoved!  
11     println!("{}", some_string)  
12 }
```

Ownership Regeln – Wann wird kopiert?

```
3 ▶ fn main() {  
4     let s1 :String = String::from( s: "hello");  
5     nimmt_ownership(s1);  
6     println!("{}", world!", s1);  
7  
8     let x :i32 = 5;  
9     macht_kopie(x);  
10    println!("{}", x);  
11  
12 }  
13  
14 1 usage  
15 fn nimmt_ownership(some_string : String){ //some_string wird gemoved!  
16     println!("{}", some_string)  
17 }  
18 1 usage  
19 fn macht_kopie(some_integer : i32) {  
20     println!("{}", some_integer)  
}
```


Ownership Regeln – Nehmen und Geben von Ownership

```
3  ▶ fn main() {
4      let s1:String = gibt_ownership();
5      println!("{}", world!", s1);
6
7      let s2:String = String::from(s: "Hello");
8      let s3:String = nimm_ownership_und_gebe_zurueck(s2); //s2 ab hier nicht mehr benutzbar!
9
10     println!("s1 = {}, s3= {}", s1, s3)
11 }
12
13 1 usage
14 fn gibt_ownership() -> String {
15     let s1: String = String::from(s: "Hello");
16     s1
17 }
18
19 1 usage
20 fn nimm_ownership_und_gebe_zurueck(some_string: String) -> String {
21     some_string
22 }
```

Ownership Regeln – Ausleihen / Borrowing

- Referenzen als Funktionsparameter zu übergeben, nennt man Borrowing

```
3 ▶ fn main() {  
4     let s1 : String = String::from( s: "Hello");  
5     let len : usize = berechne_laenge(s1);  
6     println!("Die Länge von {} ist {}.", s1, len)  
7 }  
8  
1 usage  
9 fn berechne_laenge(s: String) -> usize {  
10     let laenge : usize = s.len();  
11     laenge  
12 }
```

Ownership Regeln – Ausleihen / Borrowing

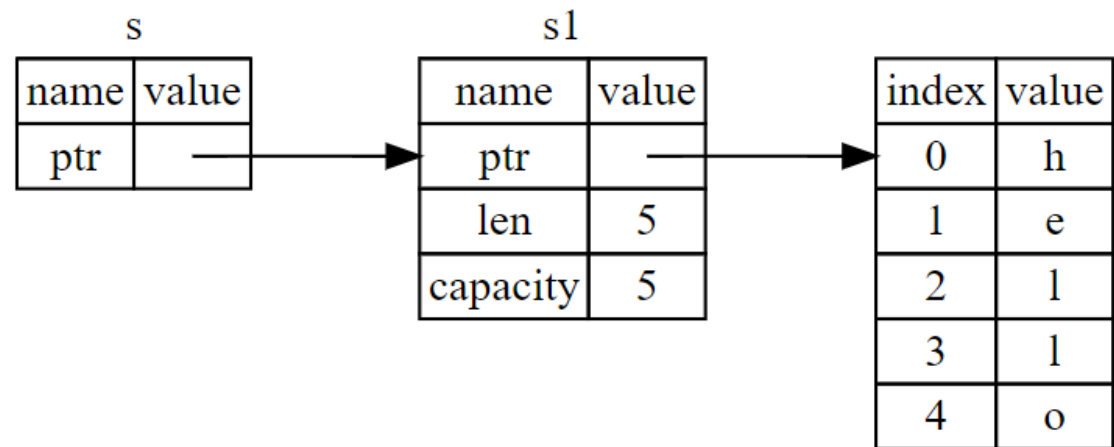
```
3 ▶ fn main() {  
4     let s1 :String = String::from(s: "Hello");  
5     let len :usize = berechne_laenge(&s1);  
6     println!("Die Länge von {} ist {}.", s1, len)  
7 }  
8  
1 usage  
9 fn berechne_laenge(s: &String) -> usize {  
10     let laenge :usize = s.len();  
11     laenge  
12 }
```

Ownership Regeln – Ausleihen / Borrowing

```

3 ▶ fn main() {
4     let s1 :String = String::from(s: "Hello");
5     let len :usize = berechne_laenge(&s1);
6     println!("Die Länge von {} ist {}. ", s1, len)
7 }
8
1 usage
9 fn berechne_laenge(s: &String) -> usize {
10     let laenge :usize = s.len();
11     laenge
12 }

```



Ownership Regeln – Ausleihen / Borrowing

Referenzen sind standardmäßig **Immutable**! Man kann sie aber mutierbar machen..

```
3 fn main() {  
4     let s1 :String = String::from( s: "Hello");  
5     veraendere_string(&s1);  
6 }  
7  
1 usage  
8 fn veraendere_string(s: &String) {  
9     s.push_str( string: ", world");  
10 }
```

Cannot borrow immutable local variable `s` as mutable

Ownership Regeln – Ausleihen / Borrowing – Veränderbare Referenz

```
3 fn main() {  
4     let mut s1: String = String::from(s: "Hello");  
5     veraendere_string(&mut s1);  
6 }  
7  
1 usage  
8 fn veraendere_string(s: &mut String) {  
9     s.push_str(string: ", world");  
10 }
```

Einschränkung! : **Maximal eine Mutable Referenz** in einem Scope
→ Um Data Races zu verhindern!

```
1 fn main() {  
2     let mut s: String = String::from("hello");  
3  
4     let r1: &mut String = &mut s;  
5     let r2: &mut String = &mut s;  
6  
7     println!("{}", r1, r2);  
8 }
```

Ownership Regeln – Ausleihen / Borrowing – Veränderbare und nicht veränderbare Referenz

```
1 fn main() {  
2     let mut s: String = String::from("hello");  
3  
4     let r1: &String = &s;  
5     let r2: &String = &s;  
6     let r3: &mut String = &mut s;  
7  
8     println!("{}", r3);  
9 }
```

&mut String

cannot borrow `s` as mutable because it is also borrowed as immutable
mutable borrow occurs here rustc(E0502)

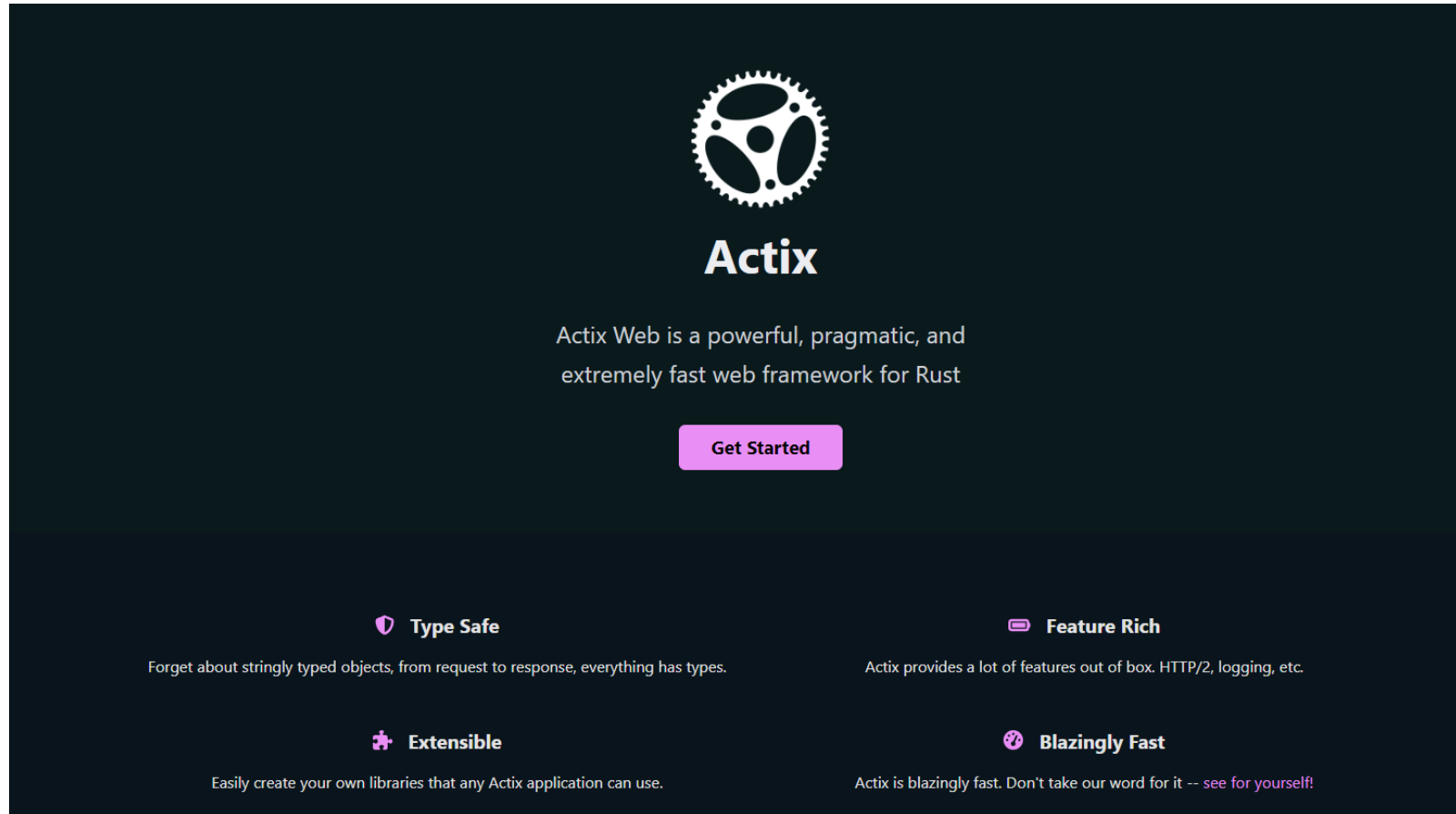
Weitere Einschränkung! : Keine mutable Referenz möglich, wenn eine immutable Referenz existiert

Ownership Regeln – Ausleihen / Borrowing – Veränderbare und nicht veränderbare Referenz

```
3 ▶ fn main() {  
4     let mut s1 :String = String::from(s: "Hello");  
5  
6     let r1 :&String = &s1;  
7     let r2 :&String = &s1;  
8  
9     println!("{}", r1,r2); // r1 und r2 werden hiernach nicht mehr benutzt, gehen out of scope  
10                             // deshalb wieder mutable referenz möglich  
11  
12     let mut r3 :&mut String = &mut s1;  
13     println!("{}", r3);  
14     veraendere_string(&mut r3);  
15 }  
1 usage  
16 fn veraendere_string(s: &mut String) {  
17     s.push_str(string: ", world");  
18 }
```


Actix-Web

Was ist Actix-Web



Quelle: <https://actix.rs/>

Wie setzt man Routen für einen Webserver mit Actix-web auf

```
#[get("/")]
async fn hello() -> impl Responder {
    HttpResponse::Ok().body("Hello world!")
}

#[post("/echo")]
async fn echo(req_body: String) -> impl Responder {
    HttpResponse::Ok().body(req_body)
}

async fn manual_hello() -> impl Responder {
    HttpResponse::Ok().body("Hey there!")
}
```

Wie setzt man einen HttpServer mit Actix-web auf

```
#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .service(hello)
            .service(echo)
            .route("/hey", web::get().to(manual_hello))
    })
    .bind(("127.0.0.1", 8080))?
    .run()
    .await
}
```

Wie setzt man einen Worker Anzahl?

```
use actix_web::{web, App, HttpResponse, HttpServer};

#[actix_web::main]
async fn main() {
    HttpServer::new(|| App::new().route("/", web::get().to(HttpResponse::Ok))).workers(4);
    // <- Start 4 workers
}
```

Wie geht man mit Path Variablen um?

```
use actix_web::{get, web, App, HttpServer, Result};

/// extract path info from "/users/{user_id}/{friend}" url
/// {user_id} - deserializes to a u32
/// {friend} - deserializes to a String
#[get("/users/{user_id}/{friend}")] // <- define path parameters
async fn index(path: web::Path<u32, String>) -> Result<String> {
    let (user_id, friend) = path.into_inner();
    Ok(format!("Welcome {}, user_id {}!", friend, user_id))
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| App::new().service(index))
        .bind(("127.0.0.1", 8080))?
        .run()
        .await
}
```

Man benutzt
Web::Path< Typ der Path Variablen >

Hier ein Tuple (u32, String)

Standard Typen werden automatisch
deserialisiert

Deserialisierung von selbst erstelltem Struct im Path

```
use actix_web::{get, web, Result};
use serde::Deserialize;

#[derive(Deserialize)]
struct Info {
    user_id: u32,
    friend: String,
}

/// extract path info using serde
#[get("/users/{user_id}/{friend}")] // <- define path parameters
async fn index(info: web::Path<Info>) -> Result<String> {
    Ok(format!(
        "Welcome {}, user_id {}",
        info.friend, info.user_id
    ))
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    use actix_web::{App, HttpServer};

    HttpServer::new(|| App::new().service(index))
        .bind(("127.0.0.1", 8080))?
        .run()
        .await
}
```

Durch das `#derive(Deserialize)` Macro, weiß actix wie es den Path deserialisieren kann

Deserialisierung von Json Data im Body

```
use actix_web::{post, web, App, HttpServer, Result};
use serde::Deserialize;

#[derive(Deserialize)]
struct Info {
    username: String,
}

/// deserialize `Info` from request's body
#[post("/submit")]
async fn submit(info: web::Json<Info>) -> Result<String> {
    Ok(format!("Welcome {}", info.username))
}
```


Immutable Application State – Am Beispiel eines Strings

```

4 struct AppState {
5     app_name: String,
6 }
7
8 // Eine Handler-Funktion, die den Application State verwendet
9 #[get("/")]
10 async fn index(data: web::Data<AppState>) -> impl Responder {
11     let app_name: &String = &data.app_name; // Zugriff auf den Application State
12     format!("Hello from {}", app_name)
13 }
14
15 #[actix_web::main]
16 async fn main() -> std::io::Result<()> {
17     // Erstelle den Application State
18     let app_state: Data<AppState> = web::Data::new(state: AppState {
19         app_name: String::from(s: "Actix Web"),
20     });
21
22     // Starte den HTTP-Server
23     HttpServer::new(move || {
24         // Anwendungskonfiguration mit dem Application State
25         App::new()
26             .app_data(app_state.clone()) : App<AppEntry> // Füge den Application State hinzu
27             .service(index)
28     })
29     .bind(addr: "127.0.0.1:8080")?
30     .run()
31     .await
32 }

```

Aufgabenvorstellung

Actix-Web API

Take Aways

- Heute haben wir folgendes kennengelernt:
 - Viele Grundlagen von Rust
 - Das Ownership Modell, welches Rust Memory Safety ermöglicht
 - Darunter Referenzen & Borrowing
- Actix-Web als Framework zum Bauen von REST API's



Traits

Tests

Async Rust (Standard Runtime, Tokio)

Lifetimes

Macros

Multithreading

Smart Pointer

Unsafe Rust

Wie könnte man sich noch weiterbilden??

- Heute haben wir folgendes kennengelernt:
 - Viele Grundlagen von Rust
 - Das Ownership Modell, welches Rust Memory Safety ermöglicht
 - Darunter auch Borrowing
 - Actix-Web als Framework zum Bauen von REST API's