



FH MÜNSTER
University of Applied Sciences

MSB

FB Wirtschaft
Münster School of Business

Rust Programming Language

Fortgeschrittenes Software Engineering

Marc Seeger (752872)
Jonathan Teige (735692)



[Bildquelle: anvilhq.com]

Agenda



Environment



Programmierkonstrukte



Algorithmische Problemstellungen



Fazit

Beispiele unter:
<https://github.com/Gersee/FSE-Rust>



[Bildquelle: Github]



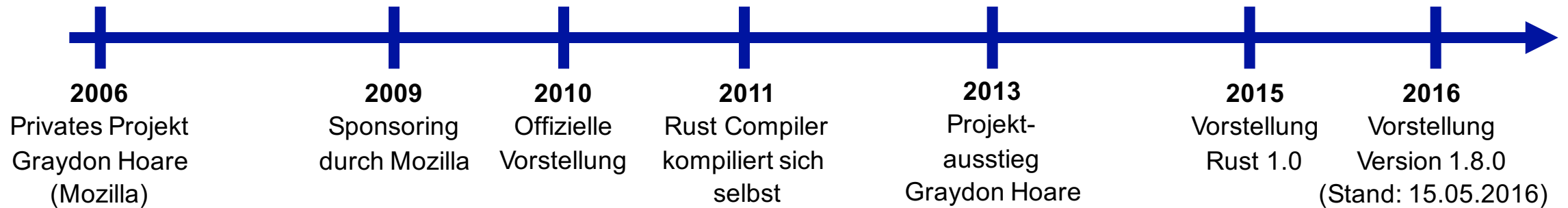
Environment

Historie, Überblick, Build, Plattform-
unterstützung, Verbreitung und Ziele



Historie

Environment



Überblick

Environment

- Systems programming language
- Multiparadigmen-Programmiersprache

Vereint Ansätze aus

- funktionaler,
- objektorientierter und
- nebenläufiger Programmierung

- Kompilierbare Sprache
- Plattformunabhängige Ausführung als Byte-Code
- Generierung von Docs unter Berücksichtigung von Markup in Kommentaren
- Opensource-Projekt mit ca. 1.400 Contributors

*„Insider-Wissen“:
Rust-Nutzer werden als
Rustaceans bezeichnet*

Build Environment

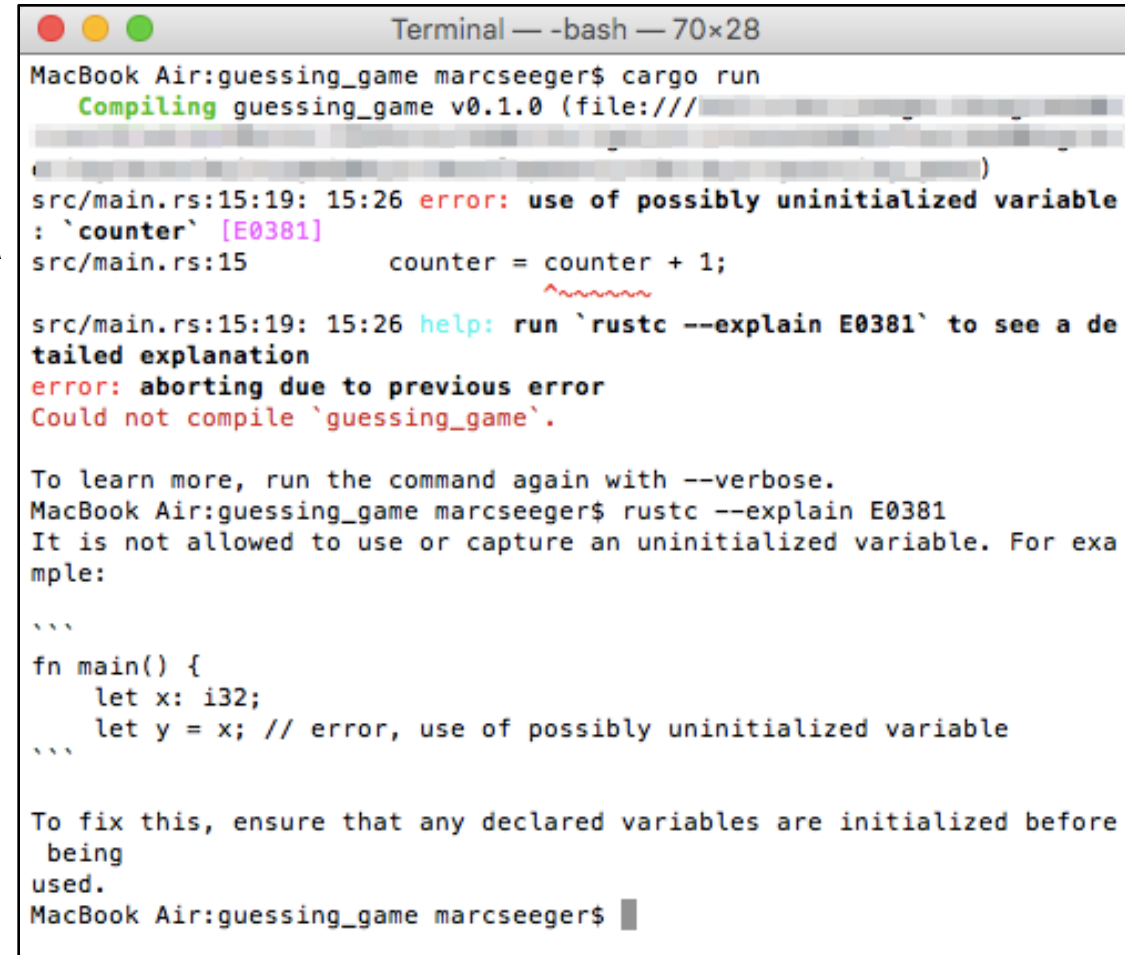
- **Cargo:** Eigenes Build- und Paketverwaltungssystem
 - Dateistrukturtemplate mit Git-Initialisierung
 - Dependency-Verwaltung
 - Fixierung der Dependencies für zukünftige Builds
 - Testdurchführung
 - Über 4.700 Crates und über 37 Mio. Downloads auf *crates.io*



[Bildquelle: crates.io]

Build Environment

- **Rustc:** Compiler
 - Verwendet *LLVM* als „Backend“
 - Einsatz von *LLVM* auch bei Ruby, C#, Swift, Scala
 - Lösungsorientierte Ausgabemeldungen
 - Warnt vor Verletzung des ***Snake-Case***



```
Terminal — -bash — 70x28
MacBook Air:guessing_game marcseeger$ cargo run
  Compiling guessing_game v0.1.0 (file:///...)
src/main.rs:15:19: 15:26 error: use of possibly uninitialized variable
: `counter` [E0381]
src/main.rs:15          counter = counter + 1;
                        ~~~~~
src/main.rs:15:19: 15:26 help: run `rustc --explain E0381` to see a de
tailed explanation
error: aborting due to previous error
Could not compile `guessing_game`.

To learn more, run the command again with --verbose.
MacBook Air:guessing_game marcseeger$ rustc --explain E0381
It is not allowed to use or capture an uninitialized variable. For exa
mple:
...
fn main() {
    let x: i32;
    let y = x; // error, use of possibly uninitialized variable
...

To fix this, ensure that any declared variables are initialized before
being
used.
MacBook Air:guessing_game marcseeger$
```

Plattformunterstützung

Environment

Unterstützte Plattformen in 32/64-Bit:

- Linux 2.6.18+
- Windows 7+ (*MSVC / MinGW*)
- Mac OS X 10.7+

Bedingte Unterstützung weiterer Plattformen

- z.B. ARM Android, PowerPC, iOS, Windows XP, Solaris

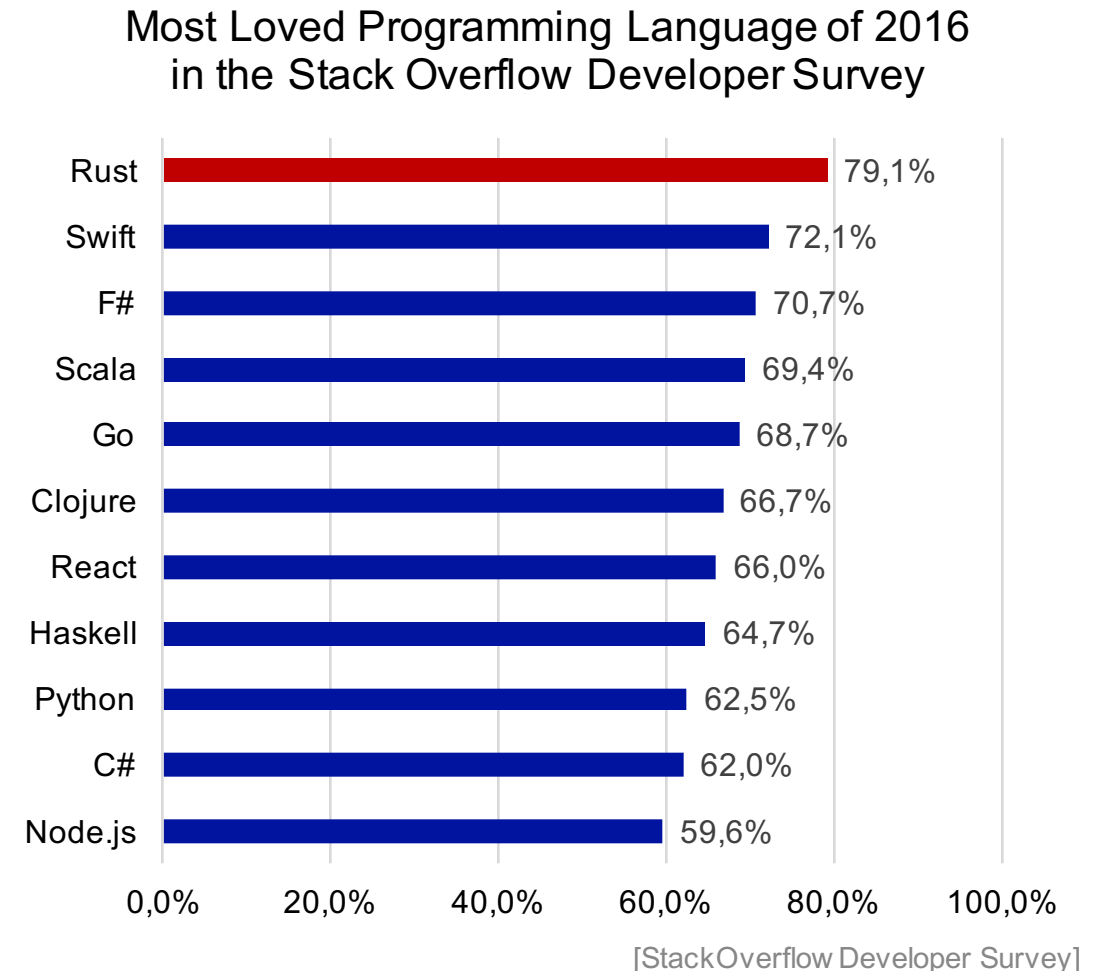
Werkzeugunterstützung durch Plugins für

- **IDE's:** Eclipse, IntelliJ IDEA, Visual Studio
- **Editoren:** Atom, Emacs, Sublime Text, Vim, Visual Studio Code

Verbreitung

Environment

- Referenzen
 - Rustc
 - Cargo
 - Servo
 - Multithreading Webbrowser Layout Engine*
 - Skylight
 - Performance Test-Framework für Ruby*
 - Redox
 - Betriebssystem*
 - Nickel.rs
 - Webapplicaton-Framework*
- *Über 4.000 Stack Overflow Questions*
- *Über 3.000 Github-Forks*



Erfahrung aus anderen Projekten

Environment



Yehuda Katz

- Ruby
- JQuery
- bpm
- ember.js



Steve Klabnik

- Swift
- npm
- Ruby



Alex Crichton

- Ruby
- homebrew
- curl



Patrick Walton

- homebrew
- doctor.js
- pdf.js

[Bildquellen: Github]

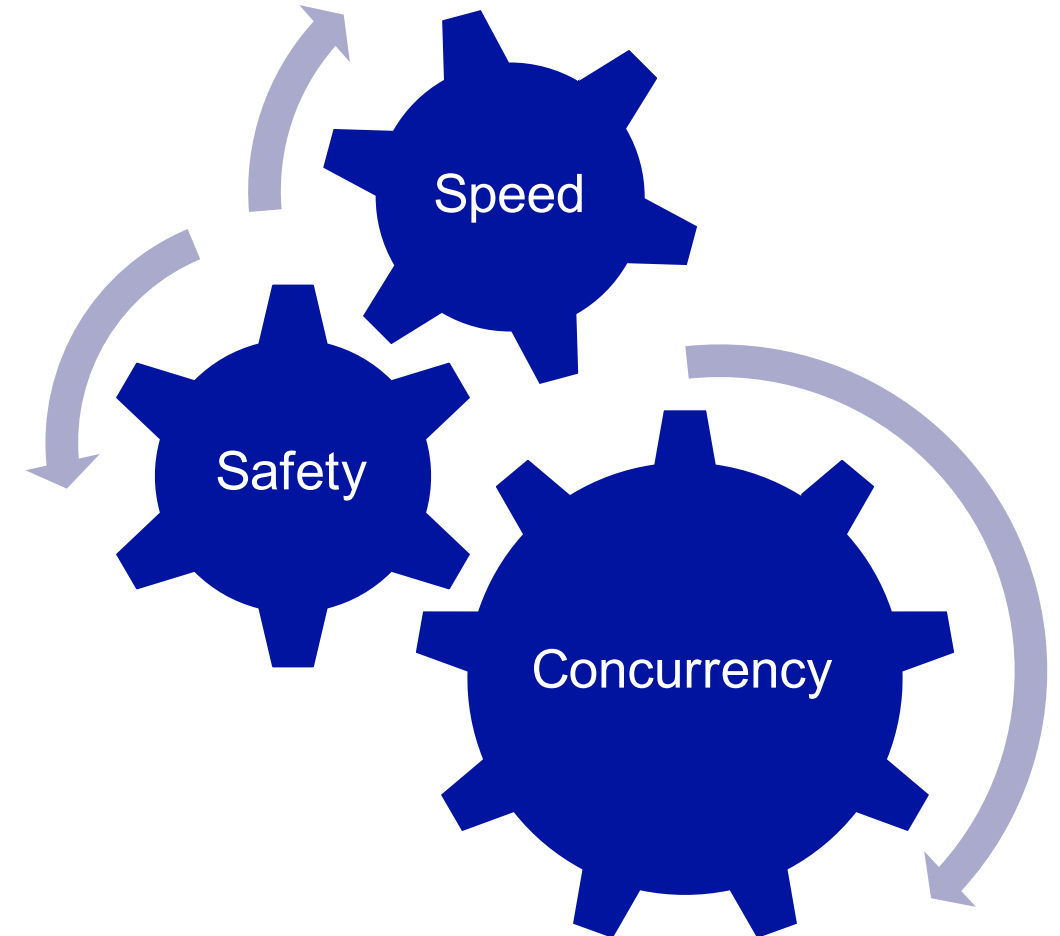
Ziele

Vorstellung



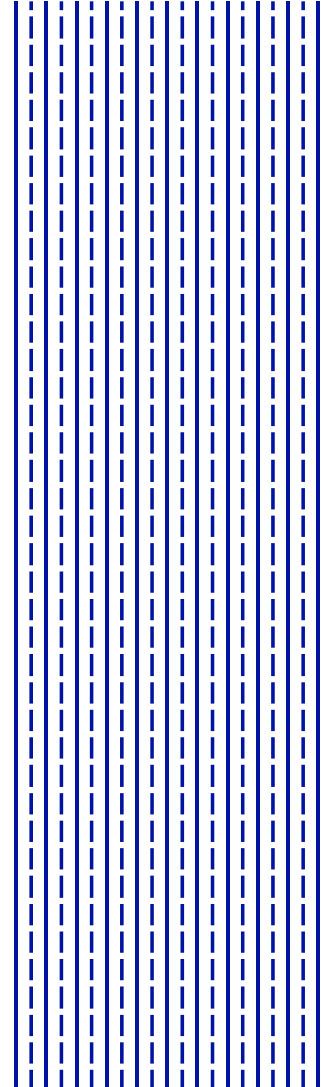
*„Rust is a systems programming language focused on three goals: **safety, speed, and concurrency**“*

- [<https://doc.rust-lang.org/book/>]





Programmier- konstrukte



```
}  
sum  
  
}  
  
fn solution2 () -> int {  
    let mut numbers = range  
    numbers.fold(0, |acc, x  
}  
  
fn solution3 () -> int {  
    use std::iter::Additiv  
    let mut numbers = rang  
    numbers.sum()  
}
```

[Bildquellen: thingiverse.com; siciarz.net]

Crates

Programmierkonstrukte

- **crate**
 - Entspricht einer Library
 - crate durch *extern*-Keyword einbinden
 - Externe crates in *Cargo.toml* angeben
- **use**
 - Entspricht der *import*-Anweisung in Java
 - Verkürzte Schreibweise beim Aufruf
 - *std*-Crate ist standardmäßig enthalten

```
1  extern crate rand;  
2  
3  use std::io;  
4  use std::cmp::Ordering;  
5  use rand::Rng;  
6
```

guessing_game: main.rs

Crates

Programmierkonstrukte

- **Cargo.toml**
 - Manifest-Datei
 - Benötigte Dependencies angegeben
 - Dynamische Angabe von Versionsnummern
- **Cargo.lock**
 - Erzeugt & Verwaltet von Cargo
 - Fixierung der exakten Dependencies für rebuild
 - Dependency-Update durch manuellen Befehl

```
1 [package]
2   name = "guessing_game"
3   version = "0.1.0"
4   authors = ["Marc Seeger <marcseeger@gmx.de>"]
5
6 [dependencies]
7   rand="*"
guessing_game: cargo.toml
```

```
1 [root]
2   name = "guessing_game"
3   version = "0.1.0"
4   dependencies = [
5     "rand 0.3.14 (registry+https://github.com/rust-lang/crates.io-index)",
6   ]
7
8 [[package]]
9   name = "libc"
10  version = "0.2.11"
11  source = "registry+https://github.com/rust-lang/crates.io-index"
12
13 [[package]]
14  name = "rand"
15  version = "0.3.14"
16  source = "registry+https://github.com/rust-lang/crates.io-index"
17  dependencies = [
18    "libc 0.2.11 (registry+https://github.com/rust-lang/crates.io-index)",
19  ]
guessing_game: cargo.lock
```


Variablen

Programmierkonstrukte

- **Typsichere** Variablendefinition → Prüfung zur Compilezeit
- „*Type Inference*“
- Standardmäßig **immutable**
- *mut* Keyword für Veränderlichkeit
- Basis-Datentypen:
 - Boolean, Char, Array, Tuple, Slice, Str
 - Numbertypes: Integer, Size, Float
- **Initialisierung** notwendig → kein Null-Pointer
- „*Primaly Expression-based-language*“

```
let secret_number = rand::thread_rng().gen_range(1, 101);  
  
let mut counter: u32 = 0;
```

guessing_game: main.rs

```
2      let x = 5;  
3  
4      let y: u8 = if x == 5 { 10 }  
5                  else { 15 };
```

Beispiel aus Rust-docs

Variablen

Programmierkonstrukte

- **Sichtbarkeit** in Blocks `{ ... }`
 - *Kein* Garbage Collector
 - out-of-scope → Ressourcenfreigabe
- **Shadowing**
- **Zugriffsbeschränkung**
 - Beliebig viele *immutable*-Referenzen
oder
 - 1 *mutable* Referenz
- Direkte Wertzuweisung macht Ursprungsvariable unbrauchbar

```
1 fn main() {  
2     let x = 5;  
3     {  
4         let x = x + 2;  
5         {  
6             let x = x * 2;  
7             println!("Inner scope: {}", x); // x=14  
8         }  
9         println!("Middle scope: {}", x); // x=7  
10    }  
11    println!("Outer scope: {}", x); // x=5  
12 }
```

shadowing: main.rs

```
let mut board: [[bool; N]; N] = [[false; N]; N];  
let mut count: i64 = 0;  
try (&mut board, 0, &mut count);
```

n_queens_problem: main.rs

Result-Type

Programmierkonstrukte

- Nutzung von **Generics**
- **Enumeration**
 - *pub enum Result<T, E>*
 - *Ok(T)*
 - *Err(E)*
- Auswertung der Ergebnisse möglich
- *expect(<msg>)* zum Abbruch im Fehlerfall

```
let guess: u32 = match guess.trim().parse() {  
    Ok(num) => num,  
    Err(_) => {  
        println!("This was not a number");  
        counter = counter - 1;  
        continue  
    },  
};
```

guessing_game: main.rs

```
io::stdin().read_line(&mut guess)  
    .expect("failed to read line");
```

guessing_game: main.rs

Komplexe Datentypen

Programmierkonstrukte

- *Keine* Klassen
 - Keine *Reflection*
 - Keine *Vererbung*
- Komplexe Datentypen durch **Structs**
 - Komposition von Attributen
 - Anonyme und benannte Attribute möglich
 - Ähneln *JSON*-Syntax
- *Struct*-Methoden
 - Durch **impl** Keyword
 - Method chaining
 - *Builder-Pattern* zur „Annäherung“ an OOP
- **Traits** garantieren Methodenimplementierung

```
1 struct NamedStruct {att_a: i32,att_b: i32,att_c: i32 }
2 struct AnonymStruct (i32, i8, i32);
3
4 trait HasAttSum {
5     fn att_sum(&self) -> i32;
6 }
7
8 impl HasAttSum for NamedStruct {
9     fn att_sum(&self) -> i32 {
10         self.att_a + self.att_b + self.att_c
11     }
12 }
13
14 fn generic_trait_use<T: HasAttSum>(stru: T) {
15     println!("Result auf att_sum method is {}", stru.att_sum());
16 }
17
18 fn main() {
19     let mut first_use = NamedStruct { att_a: 10 , att_b: 16, att_c: 20 };
20     let mut second_use = AnonymStruct(40,3,7);
21     first_use.att_b = 1200;
22     second_use.2 = 350;
23     generic_trait_use(first_use);
24 }
```

struct_trait: main.rs

Datenverarbeitung

Programmierkonstrukte

- *std*-Crate enthält **Collections**
 - Als *Structs* implementiert
 - **Sequences**: Vec, VecDeque, LinkedList
 - **Maps**: HashMap, BTreeMap
 - **Sets**: HashSet, BTreeSet
 - Misc: BinaryHeap
- Primitive Datentypen
 - **Array**
 - **Tuples** als geordnete Liste
 - **Str** als Zeichenkette, aufteilbar in *Bytes* und *Chars*
 - **Slices** als View auf Teilbereiche eines Attributes
- Ranges in Schleifen einsetzbar

```
let mut contacts = HashMap::new();

contacts.insert("Daniel", "798-1364");
contacts.insert("Ashley", "645-7689");
contacts.insert("Katie", "435-8291");
contacts.insert("Robert", "956-1745");

// Takes a reference and returns Option<&V>
match contacts.get(&"Daniel") {
    Some(&number) => println!("Calling Daniel: {}", call(number)),
    _ => println!("Don't have Daniel's number."),
}
```

Beispiel von RustByExample

Lambda & Closures

Programmierkonstrukte

- **Lambda**-Expressions werden unterstützt
 - Definieren anonyme Funktionen
 - Rückgaben möglich
 - Können an Funktionen übergeben werden
 - Pipe-getrennte *identifiers* gefolgt von *expression*
 - Expression-Angabe z.B. als Block `{...}`
- **Closures** in Verbindung mit Lambdas
 - Übergabe von Referenzen aus anderen Scope möglich
 - „*closing over*“ *the environment*

```
fn ten_times<F>(f: F) where F: Fn(i32) {  
    for index in 0..10 {  
        f(index);  
    }  
}  
  
ten_times(|j| println!("hello, {}", j));
```

Beispiel aus Rust-docs

Nebenläufigkeit

Programmierkonstrukte

- Durch *std-Crate* gegeben
- *Traits* garantieren nebenläufige Datenzugriffe
 - **Send**: Wechsel des *Owners* zwischen Threads
 - **Sync**: Threadübergreifende Referenzen
- **Thread-Programmierung** (*fork-join*)
 - Sicherheit durch Typsystem
 - **Arc<T>**: Verteilt Besitz an mehrere Threads
 - **Mutex**
- *Channels* zur Synchronisation und Kommunikation

```
let data = Arc::new(Mutex::new(vec![1, 2, 3]));

for i in 0..3 {
    let data = data.clone();
    thread::spawn(move || {
        let mut data = data.lock().unwrap();
        data[i] += 1;
    });
}
```

Beispiel aus Rust-docs

Integration von Programmiersprachen

Programmierkonstrukte

- Foreign Function Interface (FFI)
- Zugriff auf C-Bibliotheken
 - **extern** function *Wrapper*
 - **unsafe** Block
 - Zusicherung der Speichersicherheit
 - C-Compilation beim Build
- Einbindung von Rust in andere Sprachen
 - z.B. C, Python, JavaScript/ node.js, Ruby, Haskell
 - Deklaration als *pub extern fn*
 - *no_mangle* Annotation
 - *Crate* als ***staticlib*** in *Cargo.toml* typisiert

```
1 extern crate libc;
2
3 extern {
4     fn double_input(input: libc::c_int) -> libc::c_int;
5 }
6
7 fn main() {
8     let input = 4;
9     let output = unsafe { double_input(input) };
10    println!("{}", * 2 = {}, input, output);
11 }
```

c-integration: main.rs

```
1 extern crate gcc;
2
3 fn main() {
4     gcc::Config::new().file("src/double.c").compile("libdouble.a");
5 }
```

c-integration: build.rs

```
5 build = "build.rs"
6
7 [dependencies]
8 libc = "0.1"
9
10 [build-dependencies]
11 gcc = "0.3"
```

c-integration:
cargo.toml (Auszug)

Algorithmische Problemstellungen

Live-Demo

Algorithmische Problemstellungen

```
9      println!("{}", r.iter().map(|&x| if x {"x"} else {"."}.to_string()).collect::<Vec<String>>().join(" "))
10    }
11    println!("{}", count);
12    return count;
13  }
14  //Calculating algorithm
15  for i in 0..N { //column run
16    let mut ok: bool = true;
17    for j in 0..row { //row run
18      //Check if there's a conflict with an other queen
19      if board[j][i]
20        || i+j >= row && board[j][i+j-row]
21        || i+row < N+j && board[j][i+row-j]
22      { ok = false }
23    }
24    if ok {
25      //No conflict with an other queen - set position and start recursion with next row
26      board[row][i] = true;
27      try(&mut board, row+1, &mut count);
28      board[row][i] = false;
29    }
30  }
31 }
32
33 fn main() {
34   let mut board: [[bool; N]; N] = [[false; N]; N]; //board is [row][col] with a boolean flag for a queen
35   let mut count: i64 = 0;
36   try (&mut board, 0, &mut count);
37   //Reference of count given to algorithm, so it can printed out at the end
38   println!("Found {} solutions", count)
```

Projekte verfügbar unter:
<https://github.com/Gersee/FSE-Rust>



[Bildquelle: Eigene Abbildung, Github]



Fazit

```
}  
sum  
  
}  
  
fn solution2 () -> int {  
    let mut numbers = range  
    numbers.fold(0, |acc, x  
}  
  
fn solution3 () -> int {  
    use std::iter::Additiv  
    let mut numbers = rang  
    numbers.sum()  
}
```

[Bildquellen: thingiverse.com; siciarz.net]

- Ähnlichkeiten zu Java und C erkennbar
- Teilweise ungewohnte Programmierkonstrukte
- Vermeidung konkurrierender Zugriffe
- Compile-Time-Checks
- Cargo als umfangreiches Werkzeug
- Community mit Potenzial, aber verhältnismäßig klein

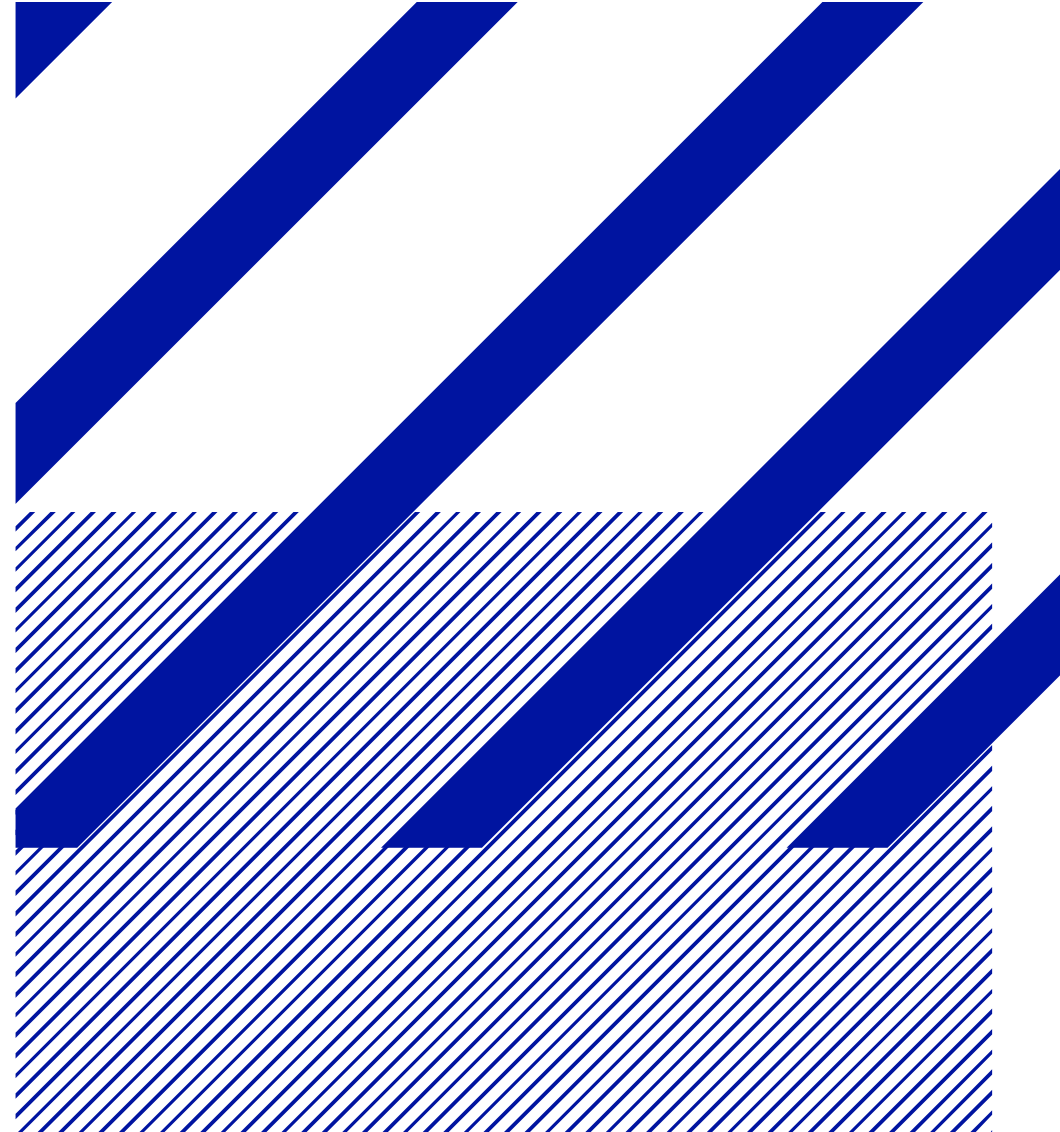
Zukünftiges Beschäftigen mit der Sprache vorgesehen!



FH MÜNSTER
University of Applied Sciences

Vielen Dank für Ihre Aufmerksamkeit!

Marc Seeger (752872)
Jonathan Teige (735692)





Anhang

```
}  
sum  
  
}  
  
fn solution2 () -> int {  
  let mut numbers = range  
  numbers.fold(0, |acc, x  
}  
  
fn solution3 () -> int {  
  use std::iter::Additiv  
  let mut numbers = rang  
  numbers.sum()  
}
```

[Bildquellen: thingiverse.com; siciarz.net]

Was bietet Rust nicht?

Anhang

- Keinen Garbage Collector
- Geringere Portierbarkeit als Java (durch natives Kompilieren)
 - Allerdings durch schlankere Laufzeitumgebung kompatibel für Ziele, welcher die JVM zu schwergewichtig ist
- Erstellung von Arrays mit Festlegung der Größe zu Laufzeit nicht möglich (bzw. nur mit Hacks)
 - Ausweichen auf Vektoren, welche eine dynamische Größe bieten
- Keine null-Werte
- Kein automatic widening coercion (in Java für primitive Typen vorhanden)
 - z.B. Integer in Float-Variable schreiben
- Kein ODBC-Treiber vorhanden
- Keine Reflection
- Keine Vererbung, dafür Traits

Was bietet Rust nicht?

Anhang

- Keine Exceptions, stattdessen panics, Results, try!()-Makros
- Kein Überladen von Methoden
- *std-Crate* von Rust wenig umfangreich
 - Häufige Notwendigkeit von Third-Party-Crates
- Schlechtere Toolunterstützung als bei etablierten Programmiersprachen

Auf Webseite angepriesene Features

Anhang

- zero-cost abstractions → Traits
- move semantics → Garantie genau einer Bindung zu jeder Ressource
- guaranteed memory safety → Ownership, Typ-Konzept
- threads without data races → Ownership, Mutex, Typ-Konzept
- trait-based generics → Generics mit Trait-Bedingung
- pattern matching → match-Operator
- type inference → Type Interference Engine (Prüfung der Initialisierungswerte und Verwendung der Variablen), explizite und implizite Typisierung
- minimal runtime → leichtgewichtige Runtime, standardmäßig wenige Crates enthalten
- efficient C bindings → Foreign Function Interface