Chemnitzer Linux-Tage

Rust als Erstsprache?

Matthias Werner

Betriebssysteme, TU Chemnitz

1. Softwarequalität

- **Beobachtung bei (studentischen) Programmier/inne/n**: Codequalität ist „verbesserungsfähig“.
- **Verschiedene Phänomene/Ursachen**:
  - Fehlendes Grundverständnis
  - Programmieren durch Ähnlichkeit
  - Vibe Coding

1. Softwarequalität

- **Beobachtung bei (studentischen) Programmier/inne/n**: Codequalität ist „verbesserungsfähig“.
- **Verschiedene Phänomene/Ursachen**:
  - Fehlendes Grundverständnis
  - Programmieren durch Ähnlichkeit
  - Vibe Coding
- **Auch Programmiersprachen tragen bei**:
  - Typenschwache Programmiersprachen wie Python/Javascript verleiten zum Quick-and-Dirty-Programmieren.
  - Sprachen wie C/C++ sind sehr fehleranfällig.

1. Softwarequalität

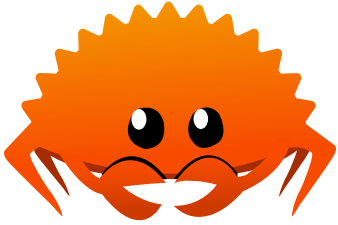
- **Beobachtung bei (studentischen) Programmier/inne/n**: Codequalität ist „verbesserungsfähig“.
- **Verschiedene Phänomene/Ursachen**:
  - Fehlendes Grundverständnis
  - Programmieren durch Ähnlichkeit
  - Vibe Coding
- **Auch Programmiersprachen tragen bei**:
  - Typenschwache Programmiersprachen wie Python/Javascript verleiten zum Quick-and-Dirty-Programmieren.
  - Sprachen wie C/C++ sind sehr fehleranfällig ➡ insb. mangelnde Speichersicherheit.

Speichersicherheit

- Zwei Drittel der Sicherheitslücken im Linux-Kernel sind auf Speichersicherheitsprobleme zurückzuführen.
- Etwa 70% aller CVEs bei Microsoft betreffen die Speichersicherheit.
- Apple-Studie: 60–70% der Sicherheitslücken in iOS und macOS sind auf Speichersicherheitslücken zurückzuführen.
- Google-Schätzung: 90% der Sicherheitslücken in Android werden durch Speichersicherheitsprobleme verursacht.
- Bei 70% aller Chrome-Sicherheitslücken handelt es sich um Speichersicherheitsprobleme.
- Einige der verbreitetsten Computerangriffe aller Zeiten wie der Slammer-Wurm, WannaCry, Trident-Exploit, HeartBleed oder Stagefright nutzen Speichersicherheitsprobleme aus.

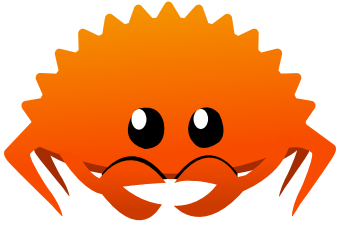
Rust

- Eine Sprache wie Rust lässt solche Fehler erst gar nicht zu.



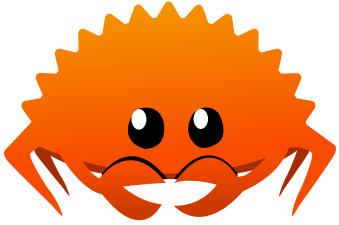
Rust

- Eine Sprache wie Rust lässt solche Fehler erst gar nicht zu.
  - Warum nicht von Anfang an nutzen?



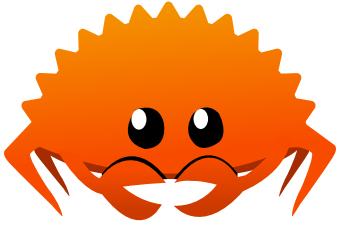
Rust

- Eine Sprache wie Rust lässt solche Fehler erst gar nicht zu.
  - Warum nicht von Anfang an nutzen?
- Rust **könnte** bieten:
  - Excellentes und schnelles Feedback durch den Compiler.
  - Sauberes Programmieren von Anfang an.



Rust

- Eine Sprache wie Rust lässt solche Fehler erst gar nicht zu.
  - Warum nicht von Anfang an nutzen?
- Rust **könnte** bieten:
  - Excellentes und schnelles Feedback durch den Compiler.
  - Sauberes Programmieren von Anfang an.
- Rust als Erstsprache?



- **Reddit-Threads**: „As my first programming language, should I learn Rust?“/„Nobody suggests learning rust as a first language.“
  - „I wouldn’t recommend it for a beginner.“
  - „I’d suggest is starting with something like Python to get a grasp of how things work and wire your mind into programming, and then move to Rust. “
  - „If people could use assembly as their first language in the past, you can certainly make it with Rust as a first language. But I definitely wouldn’t recommend it.“
  - „I don’t see anyone getting into Rust as a first language [...]“
  - „[…] I’m still hesitant to recommend Rust as a first language.“

Erstsprache Rust?

- **Bekanntes Problem**: Steile Lernkurve ➡ Man muss viel wissen, ehe man etwas machen kann.

Erstsprache Rust?

- **Bekanntes Problem**: Steile Lernkurve ➡ Man muss viel wissen, ehe man etwas machen kann.
- **Aber**: Ist Rust tatsächlich schwieriger als z. B. C++?

Erstsprache Rust?

- **Bekanntes Problem**: Steile Lernkurve ➡ Man muss viel wissen, ehe man etwas machen kann.
- **Aber**: Ist Rust tatsächlich schwieriger als z. B. C++?

Erstsprache Rust?

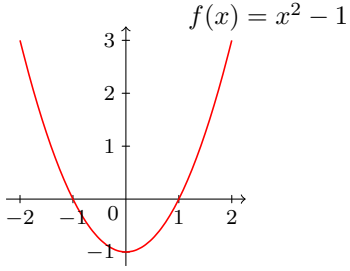
- **Bekanntes Problem**: Steile Lernkurve ➡ Man muss viel wissen, ehe man etwas machen kann.
- **Aber**: Ist Rust tatsächlich schwieriger als z. B. C++?
- Rust ist auf jeden Fall **anders** ➡ vielleicht eine andere Didaktik notwendig?

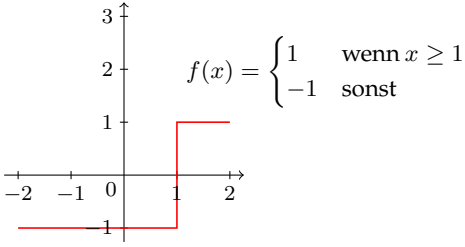
Erstsprache Rust?

- **Bekanntes Problem**: Steile Lernkurve ➡ Man muss viel wissen, ehe man etwas machen kann.
- **Aber**: Ist Rust tatsächlich schwieriger als z. B. C++?
- Rust ist auf jeden Fall **anders** ➡ vielleicht eine andere Didaktik notwendig?
- **Probe**: Kurs „**Algorithmen und Programmierung**“
  - 10 LP
  - Ca. 150 Teilnehmer pro Semester
  - Elf verschiedene Studiengänge
  - Überwiegend erstes Semester Bachelor, aber auch Master
  - Z. T. schon Vorkenntnisse in Programmierung (privat oder Schule), überwiegend Python/Javascript (ca. 50%)
  - Bisherige Sprache: C

2. Didaktik

Funktion

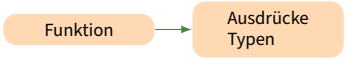


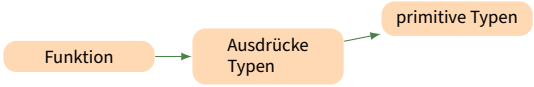



```rust
fn function(x: f32) -> f32 {
  x * x -1
}
```

```rust
fn function(x: f32) -> f32 {
  if x >= 1 { 1 } else { -1 }
}
```

Funktion


Funktion
Ausdrücke
Typen


Funktion
Ausdrücke
Typen
primitive Typen

Was ist ein Typ?

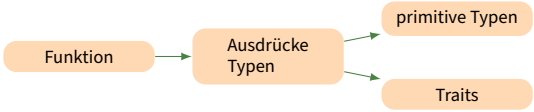
Ein **Typ** beschreibt…

- die rechnerinterne Darstellung des Datums;
- welche Werte das Datum annehmen kann;
- wie sich das Datum verhalten kann bzw. welchen Operationen es ausgesetzt werden kann;
- was das Datum repräsentiert bzw. wofür es genutzt werden soll.

Was ist ein Typ?

Ein **Typ** beschreibt…

- die rechnerinterne Darstellung des Datums;
- welche Werte das Datum annehmen kann;
- wie sich das Datum verhalten kann bzw. welchen Operationen es ausgesetzt werden kann;
- was das Datum repräsentiert bzw. wofür es genutzt werden soll.

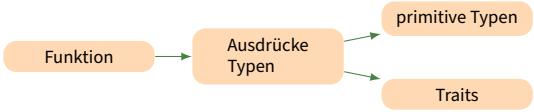

Funktion
Ausdrücke
Typen
primitive Typen
Traits

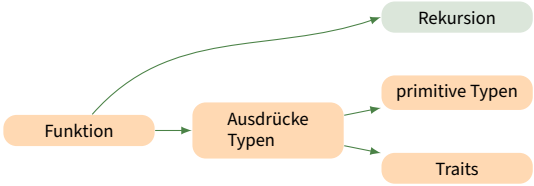
- „Trait“ im Wortsinne auffassen: Eigenschaft, Merkmal, (Verhaltens-)charakteristik

- „Trait“ im Wortsinne auffassen: Eigenschaft, Merkmal, (Verhaltens-)charakteristik
  - Mit Markertraits anfangen.

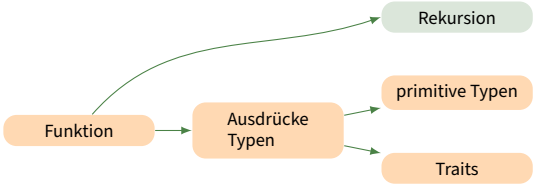
- „Trait“ im Wortsinne auffassen: Eigenschaft, Merkmal, (Verhaltens-)charakteristik
  - Mit Markertraits anfangen.
  - Verhalten (Methoden und assozierte Funktionen) als Verallgemeinerung.

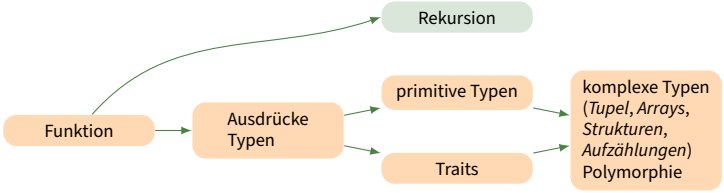
- „Trait“ im Wortsinne auffassen: Eigenschaft, Merkmal, (Verhaltens-)charakteristik
  - Mit Markertraits anfangen.
  - Verhalten (Methoden und assozierte Funktionen) als Verallgemeinerung.
  - Am Anfang nur `self`-Methoden ➡ primitive Typen sind alle `Copy`.

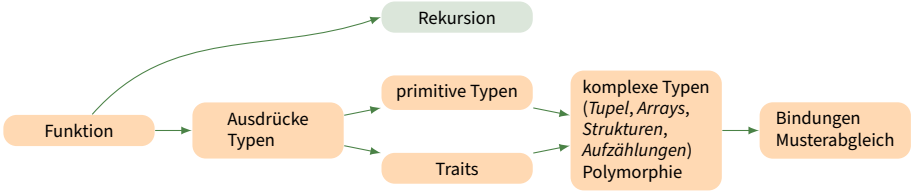

Funktion
Ausdrücke
Typen
primitive Typen
Traits

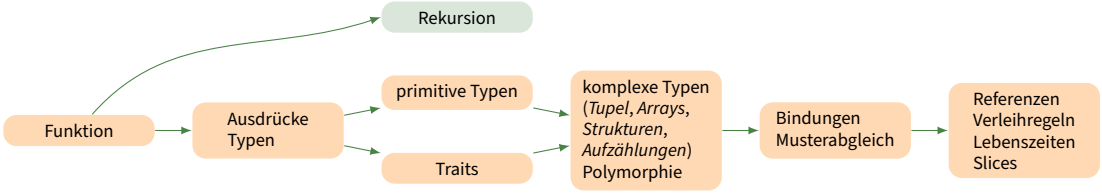

Rekursion
primitive Typen
Funktion
Ausdrücke
Typen
Traits

- Klassische „Schulberechnungen“ sind umsetzbar, z. B.:
  - Größter gemeinsamer Teiler (Euklidscher Algorithmus)
  - Flächen und Volumenberechnungen
  - Physikaufgaben wie Weg-Zeit- oder Frequenz-Wellenlängen-Bestimmung
  - Lösung quadratischer Gleichungen
- Ohne Zustände (Variablen) und `Copy`-Typen keine Probleme mit Borrowchecker etc.
- Übergabe von (Zahlen-)Werten auf der Kommandozeile wird als „Blackbox“-Modul zur Verfügung gestellt.


Rekursion
primitive Typen
Funktion
Ausdrücke
Typen
Traits


Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie

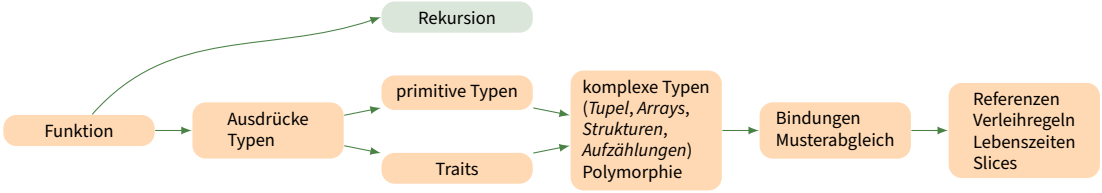

Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich

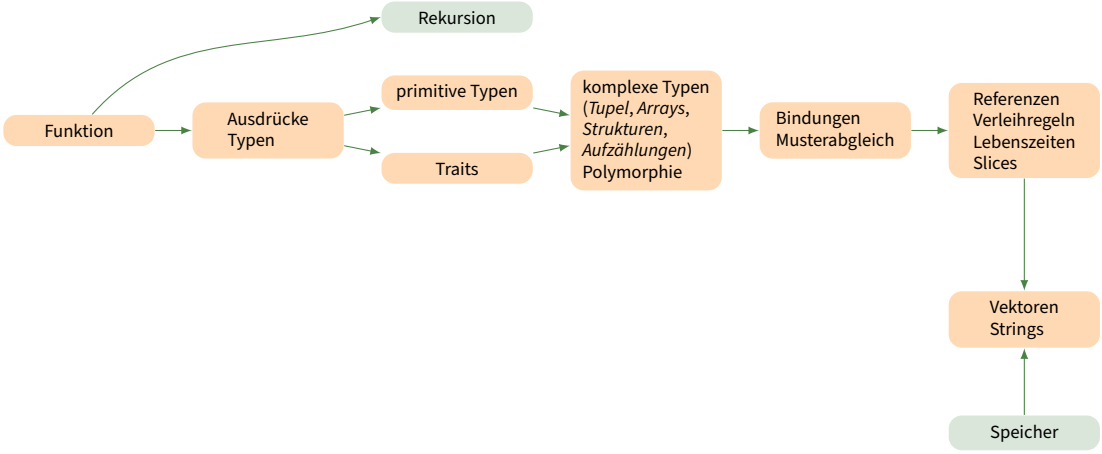

Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices

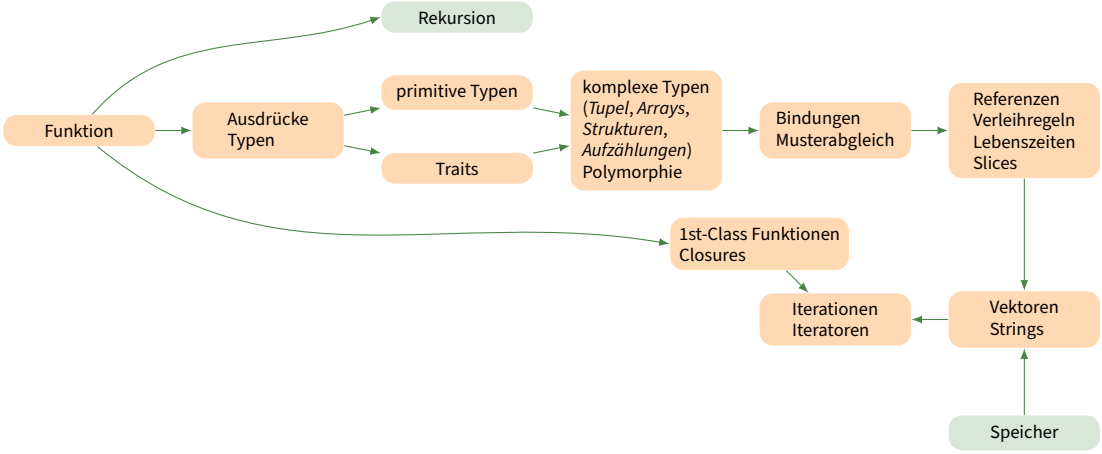
Merke

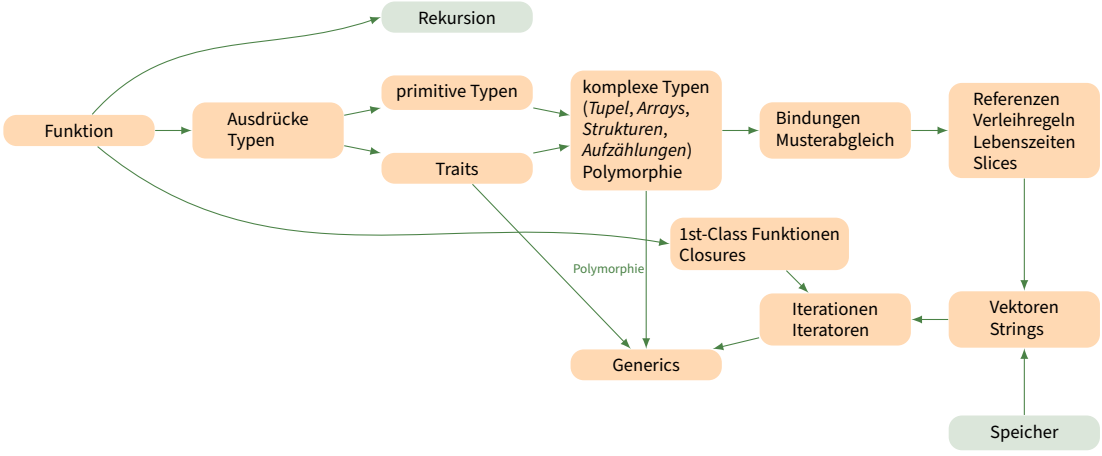
Die zeitnahe Einführung von Referenzen und Lebenszeiten zu Bindungen (Variablen) lässt keine Zeit für die Etablierung schlechter Angewohnheiten.

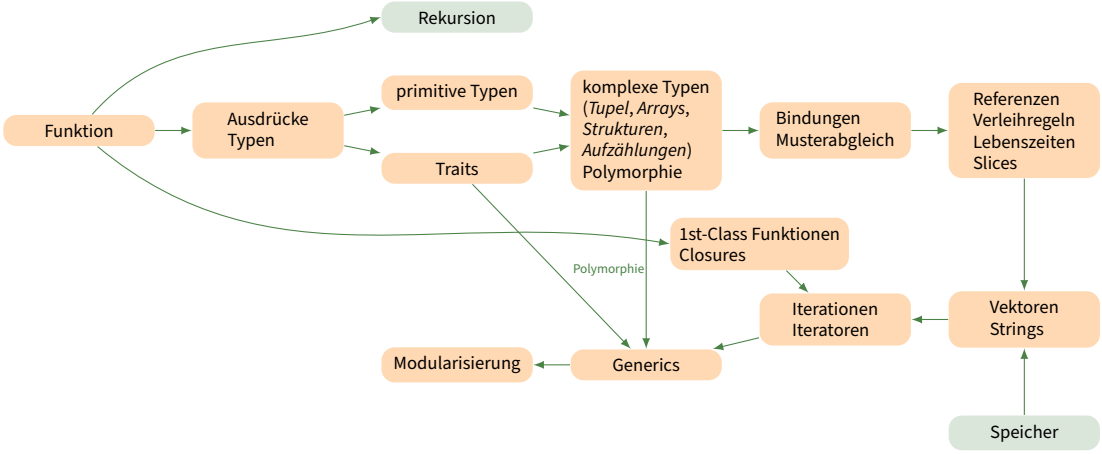
Lebenszeitannotationen werden als Hilfestellung für den Compiler „organischer“ wahrgenommen.

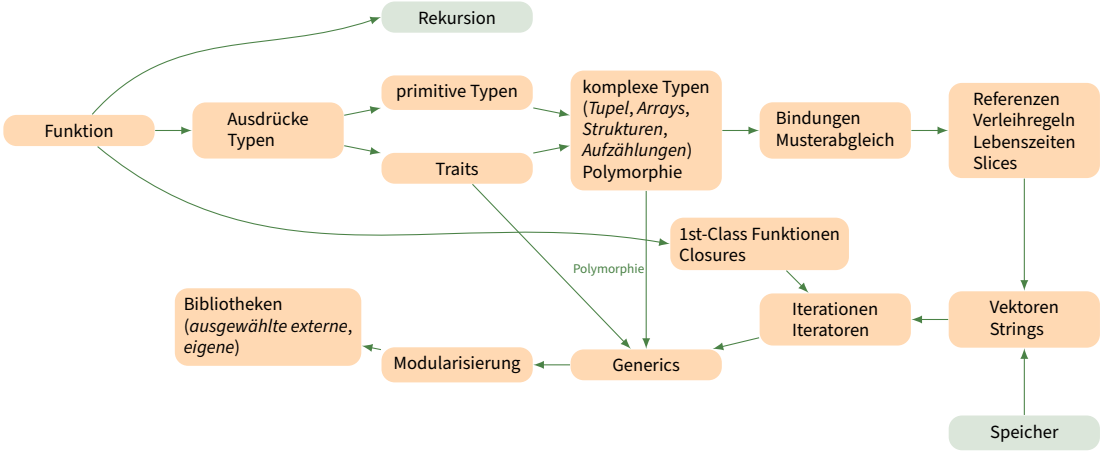

Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices

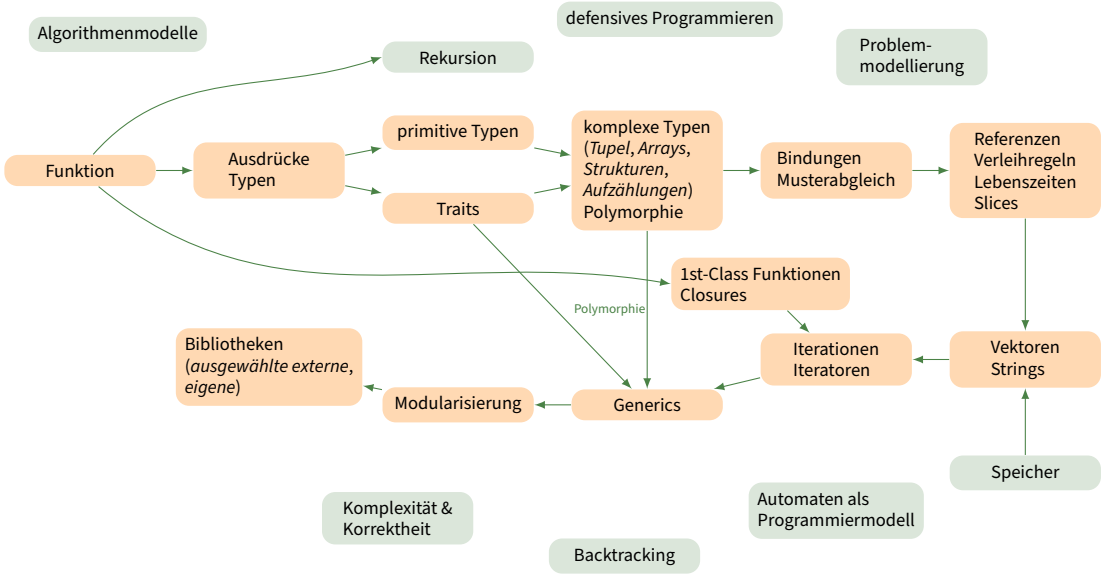

Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices
Vektoren
Strings
Speicher

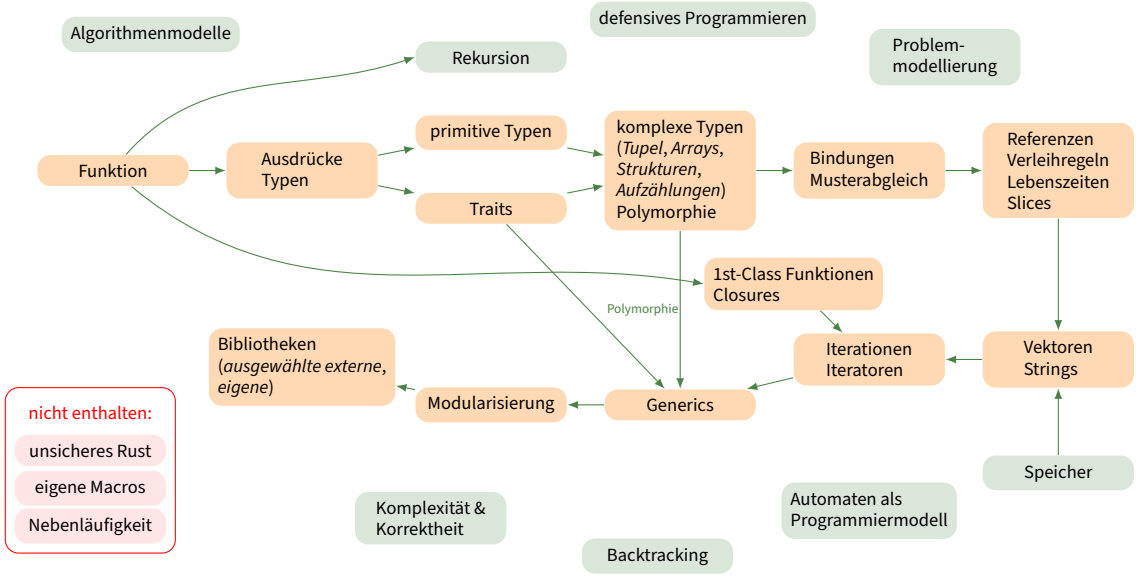

Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices
1st-Class Funktionen
Closures
Iterationen
Iteratoren
Vektoren
Strings
Speicher


Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices
1st-Class Funktionen
Closures
Polymorphie
Iterationen
Iteratoren
Vektoren
Strings
Generics
Speicher


Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*, *Strukturen*, *Aufzählungen*)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices
1st-Class Funktionen
Closures
Polymorphie
Iterationen
Iteratoren
Vektoren
Strings
Modularisierung
Generics
Speicher


Rekursion
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*, *Strukturen*, *Aufzählungen*)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices
1st-Class Funktionen
Closures
Polymorphie
Iterationen
Iteratoren
Vektoren
Strings
Bibliotheken
(*ausgewählte externe*, *eigene*)
Modularisierung
Generics
Speicher


Algorithmenmodelle
defensives Programmieren
Rekursion
Problem-
modellierung
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices
1st-Class Funktionen
Closures
Polymorphie
Bibliotheken
(*ausgewählte externe*,
eigene)
Iterationen
Iteratoren
Vektoren
Strings
Modularisierung
Generics
Speicher
Komplexität &
Korrektheit
Automaten als
Programmiermodell
Backtracking


Algorithmenmodelle
defensives Programmieren
Rekursion
Problem-
modellierung
Funktion
Ausdrücke
Typen
primitive Typen
Traits
komplexe Typen
(*Tupel*, *Arrays*,
Strukturen,
Aufzählungen)
Polymorphie
Bindungen
Musterabgleich
Referenzen
Verleihregeln
Lebenszeiten
Slices
1st-Class Funktionen
Closures
Polymorphie
Bibliotheken
(*ausgewählte externe*,
eigene)
Iterationen
Iteratoren
Vektoren
Strings
Modularisierung
Generics
nicht enthalten:
unsicheres Rust
eigene Macros
Nebenläufigkeit
Speicher
Komplexität &
Korrektheit
Automaten als
Programmiermodell
Backtracking

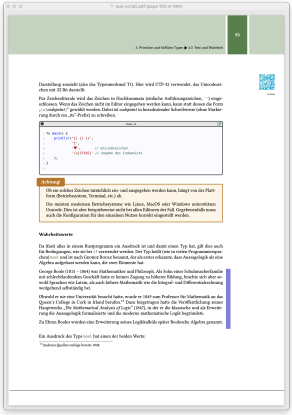
3. Lehrform

- Verzicht auf klassische Vorlesung ➡ Lehrmodell an **Flipped Classroom** angelehnt

3. Lehrform

- Verzicht auf klassische Vorlesung ➡ Lehrmodell an **Flipped Classroom** angelehnt
- Ausführliches Vorlesungsskript mit
  - interaktiven Codebeispielen
  - farblich markierten Zusatzinformtionen
  - weiterführenden Links (QR)

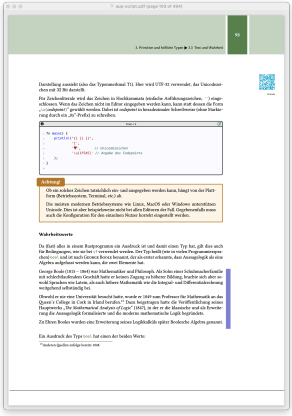
  zur Vorbereitung



3. Lehrform

- Verzicht auf klassische Vorlesung ➡ Lehrmodell an **Flipped Classroom** angelehnt
- Ausführliches Vorlesungsskript mit
  - interaktiven Codebeispielen
  - farblich markierten Zusatzinformtionen
  - weiterführenden Links (QR)

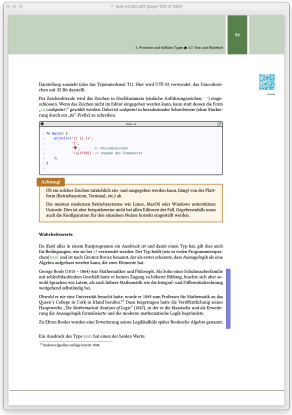
  zur Vorbereitung
- „**Konzeptdiskussion**“ statt Vorlesung zur Klärung von Schwierigkeiten und Edge Cases



3. Lehrform

- Verzicht auf klassische Vorlesung ➡ Lehrmodell an **Flipped Classroom** angelehnt
- Ausführliches Vorlesungsskript mit
  - interaktiven Codebeispielen
  - farblich markierten Zusatzinformtionen
  - weiterführenden Links (QR)

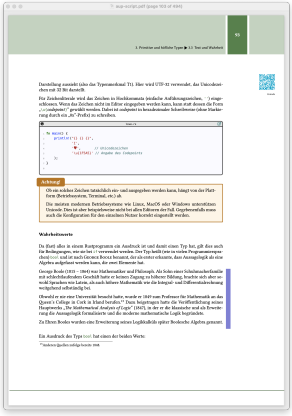
  zur Vorbereitung
- „**Konzeptdiskussion**“ statt Vorlesung zur Klärung von Schwierigkeiten und Edge Cases
- praktische Übungen am Rechner



3. Lehrform

- Verzicht auf klassische Vorlesung ➡ Lehrmodell an **Flipped Classroom** angelehnt
- Ausführliches Vorlesungsskript mit
  - interaktiven Codebeispielen
  - farblich markierten Zusatzinformtionen
  - weiterführenden Links (QR)

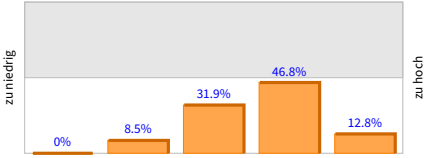
  zur Vorbereitung
- „**Konzeptdiskussion**“ statt Vorlesung zur Klärung von Schwierigkeiten und Edge Cases
- praktische Übungen am Rechner
- semesterbegleitende Prüfung (Programmierbelege)



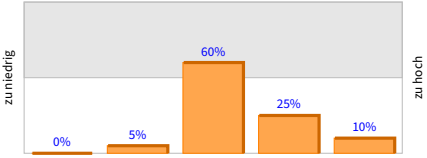
4. Evaluation

2020/21 (C)

- Die Anforderungen der Veranstaltung an mich waren…



2024/25 (Rust)

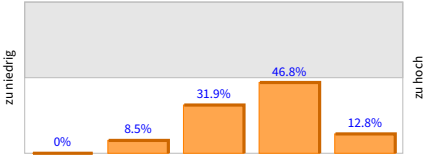


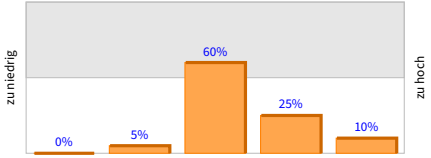
4. Evaluation

2020/21 (C)

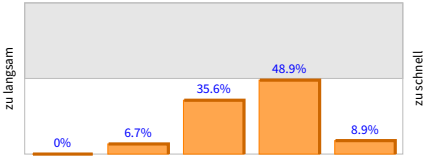
2024/25 (Rust)

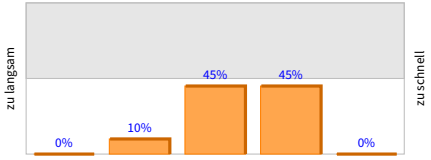
- Die Anforderungen der Veranstaltung an mich waren…





- Das Tempo der Vorlesung war für mich…



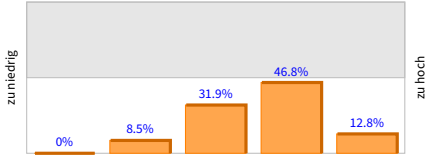


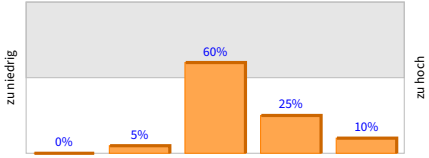
4. Evaluation

2020/21 (C)

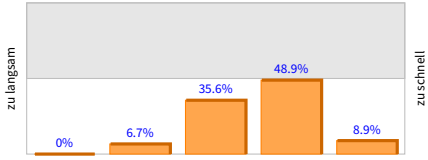
2024/25 (Rust)

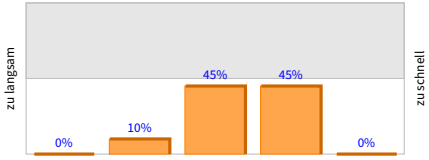
- Die Anforderungen der Veranstaltung an mich waren…





- Das Tempo der Vorlesung war für mich…





- Insgesamt würde ich der Veranstaltung folgende Schulnote geben:
  Durchschnitt: $1{,}77$ Durchschnitt: $1{,}36$

Evaluation (Forts.)

- Befragung nach größten Schwierigkeiten
  - „Borrowchecker“
  - „Unterschiede zu anderen Sprachen“
  - „Generics“
  - „Lifetimes“
  - „Referenzen“
  - „Verwendung von Traits“
  - „Rekursion“
  - „Konzept von Speicher“

Evaluation (Forts.)

- Befragung nach größten Schwierigkeiten **(nur Erstprogrammierer)**
  - ~~„Borrowchecker“~~
  - ~~„Unterschiede zu anderen Sprachen“~~
  - ~~„Generics“~~
  - ~~„Lifetimes“~~
  - ~~„Referenzen“~~
  - „Verwendung von Traits“
  - „Rekursion“
  - „Konzept von Speicher“

Erkenntnis/These

Die Schwierigkeit von Rust liegt **nicht** (in erster Linie) in seinen eigenen Sprachkonstrukten, sondern vor allem in der Betrachtung aus der Perspektive **anderer Programmiersprachen**.

Erkenntnis/These

Die Schwierigkeit von Rust liegt **nicht** (in erster Linie) in seinen eigenen Sprachkonstrukten, sondern vor allem in der Betrachtung aus der Perspektive **anderer Programmiersprachen**.

> *„A language that doesn't affect the way you think about programming is not worth knowing.“*
>
> – Alan J. Perlis

- Bei Interesse: Materialien (Skript, Slides) stehen unter CC
- `https://gitlab.hrz.tu-chemnitz.de/osg/aup-script`

Fragen?

Kommentare?