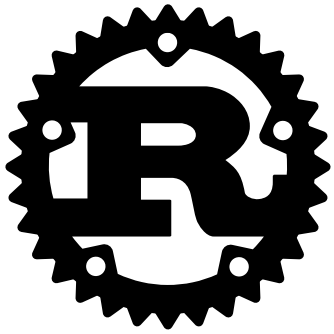

Programmieren mit Rust

2019 ist Rust von den Entwicklern bei der jährlichen *Stack Overflow*-Umfrage das vierte Jahr in Folge zur beliebtesten Programmiersprache gewählt worden.[^1] Diese Beliebtheit unter Entwicklern kommt nicht von ungefähr. Rust verbindet eine Reihe von Eigenschaften, die es von anderen Sprachen deutlich absetzt und so für seine einzigartige Stellung im Ökosystem der Programmiersprachen sorgt. Allen voran ermöglicht der innovative Ansatz des "*Borrow Checking*" robuste und sichere Programme zu schreiben, ohne einen Leistungsverlust zur Laufzeit dafür in Kauf nehmen zu müssen, wie es überlicherweise bei speicherverwalteten Sprachen (*Garbage Collection*), z.B. Go, der Fall ist. Die Möglichkeit, zugleich genauso verlässliche wie sehr performante Programme zu schreiben, macht Rust damit seit Jahren zum ersten wirklich ernstzunehmenden "*Herausforderer*" von C als Systemsprache. So wird beispielsweise derzeit darüber diskutiert, Komponenten des Linuxkernels auch in Rust zuzulassen.[^2] Zusätzlich zeigen diverse Benchmarks im direkten Vergleich auch eine erhebliche Effizienz zu alternativen Programmiersprachen, indem Rust sich neben der Stabilität auch durch die effiziente Programmierung abhebt.[^3][^4] Rust verwendet LLVM und ist daher auch auf allen verwendeten Computerarchitekturen bzw. Plattformen vom Raspberry Pi (*ARM*) bis hin zum Großrechner (*s390*) ausführbar.[^5]

Dadurch, dass Rust moderne abstrakte High-Level-Konzepte integriert, ist es neben der Systemprogrammierung auch für ein breit gefächertes Anwendungsfeld interessant, das von verteilten Systemen über Web-Anwendungen - Rust unterstützt WASM/WebAssembly direkt und läuft im Browser - und Desktop-Applikationen bis hin zu Data-Science-Anwendungen reicht. Nahezu alle "*Großen*" der IT-Branche, wie z.B. Amazon, Google, Microsoft, Dropbox, Mozilla etc. setzen Rust ein und haben Projekte, in denen sich Rust bereits praktisch bewährt.

Ziel:

Die Vorlesung soll eine grundlegende Einführung in Rust an sich und die darin verwirklichten Konzepte sowie Paradigmen geben. Ein besonderes Augenmerk wird dabei auf die Vermittelung des einzigartigen *Life-Time / Ownership / Borrowing*-Management gelegt. Anschließend sollen die erlernten Konzepte in Programmier-Projekten aus dem ganzen Anwendungsbereich von Rust vertieft werden, indem die Vorteile von Rust gezeigt werden. Dadurch soll die Verwendung von Rust in anderen Projekten ermöglicht werden.

Themen:

- Rust ganz kurz - Geschichte, Einordnung als Programmiersprache
- Die Arbeitsumgebung und Ecosystem - Build-Tool chain *cargo* , Editor/IDE
- Grundlegende Programmkonstrukte - Variablen, einfache Datentypen, Funktionen, Kontrollfluss

- Arten von Strings in Rust *str* / *String*
- Fehlerbehandlung - *Result* / *Option*
- Der *Ownership*-Mechanismus - *Life-Time* / *Ownership* / *References* / *Borrowing*
- Komplexe Datenstrukturen, *Enums*, Pattern-Matching
- Management für große Projekte - *Packages*, *Module*, *Creates*
- Collections-Types
- Error-Types
- Generische Typen, *Traits*
- Automatisierte Tests
- Funktionale Konzepte - *Futures*
- Smart Pointer
- Concurrency
- Projekte aus den Einsatzbereichen von Rust wie Verteilte Systeme, Data-Science und Web-Anwendungen

Literatur:

- "The Rust Programming Language"
- T.S. McNamara: "Rust in Action", Manning Publications (in MEAP)
- J. Blandy, J. Orendorff: "Programming Rust - Fast, Safe Systems Development", O'Reilly Media

Links:

- [^1]: Stack Overflow Annual Developer Survey 2019 - Most Loved, Dreaded, and Wanted
- [^2]: Rust is the future of systems programming, C is the new Assembly (Packt)
- [^3]: The Computer Language Benchmarks Game
- [^4]: Web Framework Benchmarks - 2019-07-09 - techempower.com
- [^5]: Rust Plattform Support
- Rust Homepage