



Einführung in Rust

Moderne Programmierung für die nächste Generation

Andreas Wallner

2025-04-09



Ich?



Seit > 10 Jahren bei Infineon Technologies

Principal Engineer Contactless Innovation / Software & Firmware

In "Contactless Innovation" Team als Design Lead

Pre-Tapeout: System / Digital Design

Post-Tapeout: Software

Unregelmäßig in Produktentwicklung

In anderen Teams als Feuerwehr 🚒

Verbesserung von Entwicklungsflows

Rust für open-source / privat / Infineon

Organisatorisches

- Zeit 09:00 – 16:30
- Mittagspause: ~12:15, Tisch Rondo
- Kaffeepausen: wenn es reinpasst
- Fotos?
- Fragen: immer gleich
- Englisch / Deutsch
- Links
- https://tlk.io/rust_training
- DANKE AN GOOGLE: <https://google.github.io/comprehensive-rust>

Das Program

- 1. Intro
- 2. Variablen
- 3. Blöcke
Datenfluss
- 4. Tuples
Structs
Enums

- 1. Pattern matching
Fehlerbehandlung
- 2. Referenzen
Lifetimes
- 3. Iteratoren
- 4. Smart Pointer
- 5. Reflection / Outlook

Intro

Umfrage Programmiersprachen



Rust

- Programmiersprache
- nativ / compiliert
- statisch, stark, linear, inferiert typisiert

- 2008 – Hobbyprojekt von Graydon Hoare
- 2010 – Mozilla
- 2015 – Erste Alpha Release
- 2020 – Rust Foundation



Warum Rust?

70%

<https://www.chromium.org/Home/chromium-security/memory-safety/>

<https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/>

How Cyber Thieves Use Your Smart Fridge As Door to Your Data

Published Jun 23, 2021 at 4:00 PM EDT

Updated Jun 23, 2021 at 4:04 PM EDT

By [Alex J. Rouhandeh](#)
Special Correspondent

FOLLOW

by Michael Kan
U.S. Correspondent

Smart teddy bears involved in a contentious data breach

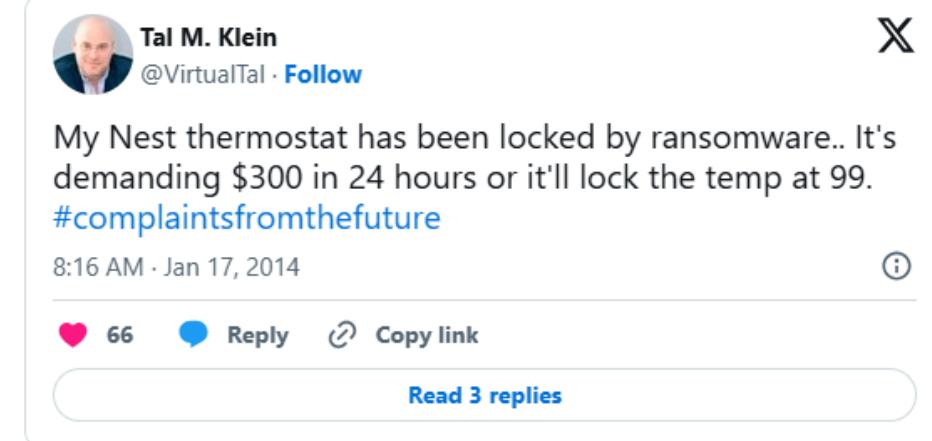
News
28 Feb 2017 • 4 mins

arXiv:2306.09017 (cs)

[Submitted on 15 Jun 2023 (v1), last revised 26 Jul 2023 (this version, v3)]

Who Let the Smart Toaster Hack the House? An Investigation into the Security Vulnerabilities of Consumer IoT Devices

Yang Li, Anna Maria Mandalari, Isabel Straw



Tal M. Klein (@VirtualTal) · Follow X

My Nest thermostat has been locked by ransomware.. It's demanding \$300 in 24 hours or it'll lock the temp at 99. #complaintsfromthefuture

8:16 AM · Jan 17, 2014

1,130 Retweets 66 Likes

Reply Copy link

Read 3 replies

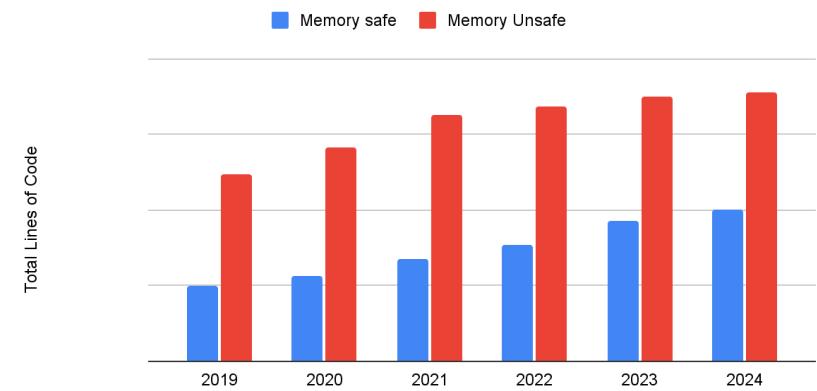
Akzeptanz in Industrie

- Verwendung stetig ansteigend (2023 ~1.45%, TIOBE Platz 13)
- Microsoft: 'Best Chance' at Safe Systems Programming (Video)
- Google: AOSP wechselt langsam auf Rust (2021, 2022, 2024)
- Amazon: Prime Video UI in Rust, Rust in EC2 & S3

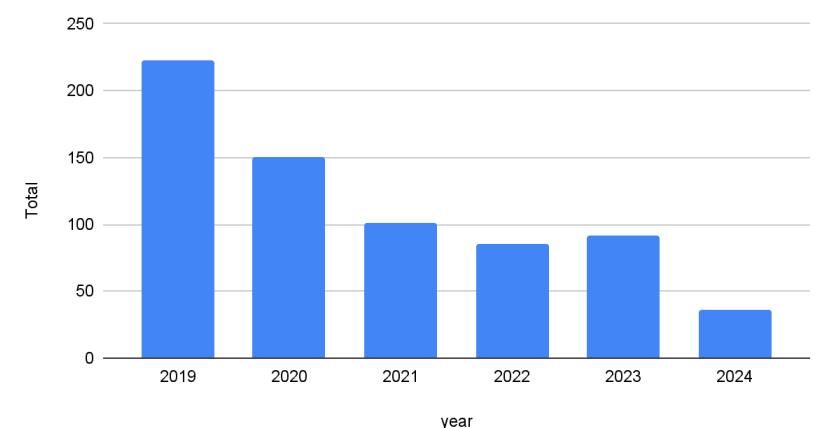
- Volvo ECU für Elektroautos in Rust
- Infineon Automotive uCs mit Rust support (inklusive Aurix compiler)
- Elektrobit: AUTOSAR kompatible „basic software“

- EU IoT: Cybersecurity Guideline
 - memory safety noch nicht vorgeschrieben
- Memory-safe languages empfohlen/verlangt von
 - US CISA, NSA, FBI, AUS CSC, CAN CCS, UK CSC, NZ CSC
- Code aktiv nach Rust portieren: DARPA TRAKTOR

Total Memory safe and Memory Unsafe Lines of Code in AOSP



Number of Memory Safety Vulns per Year



Quelle: [Google](#)

Safe Rust

kein “Undefined Behavior”

lesen/schreiben außerhalb Allokationen

use-after-free

data races

Uninitialisierte Variablen verwenden

Null-Pointer dereferenzieren

Zugriff auf nicht aktive Union-Member

(Ganzzahl Overflow)

...

nicht: Programmabbruch

<https://doc.rust-lang.org/stable/reference/behavior-considered-undefined.html>

Moderne Sprache

- Spagat zwischen low-level und high-level
- Namespaces, UTF-8 Strings, ...
- Große Standardbibliothek
- Multithreaded / Asynchrone Programme
- Pakete / Module / Paketeverwaltung
 - expliziter Export / Sichtbarkeit
- Ökosystem
 - Cargo
 - crates.io / docs.rs / lib.rs
 - rustfmt
 - rustdoc
 - clippy
 - test
 - ...



Setup

- <https://rustup.rs/>
- cargo new <project> / cargo new --lib <project>

```
% tree
├── Cargo.toml
└── bin
    └── other_binary.rs
└── src
    ├── main.rs
    └── module.rs
```

main test

```
fn main() {
    println!("Hello, world!");
}

#[cfg(test)]
mod tests {
    #[test]
    fn it_works() {
        assert_eq!(2 + 2, 4);
    }
}
```

Hands-on 01

Clonen / Downloaden / Unzippen der Übungsbeispiele

```
cargo run --bin rust-training  
cargo test --workspace
```

Blick auf die Filestruktur
Gemeinsam: modules
Hello world & “Hello test”

Variablen, Arithmetische Datentypen und Mutabilität

Variablen

- `let` deklariert variablen -> unveränderbar, defaultd
- `let mut` deklariert veränderbare variablen
- `const` deklariert constanten (wert bekannt beim compilieren)
- redeklaration ist möglich

```
fn main() {  
    let x: i32 = 10;  
    println!("x: {}", x);  
    // x = 20;  
    // println!("x: {}", x);  
}
```

<https://doc.rust-lang.org/book/ch03-01-variables-and-mutability.html>

Numerische Datentypen

	Typen	Werte
Signed	i8, i16, ..., i128, isize	-42, 0, 1_123, 456_i64
Unsigned	u8, u16, ..., u128, usize	50, 0b101, 0xaffe, 10u16
Float point	f32, f64	2.71, 1.2e3, 2_f32
Unicode Buchstabe	char	'a', '☺', 'だ'
Bool	bool	true, false

Zeichen sind Unicode (UTF-8) Codepoints, keine graphemes

Arithmetik

- Wie in anderen Sprachen, im debug Modus überprüft

```
fn calculation(a: i32, b: i32, c: i32) -> i32 {
    return a * b + b * c + c * a;
}

fn safe_calculation(a: i32, b: i32, c: i32) -> i32 {
    return (a * b).saturating_add(b * c).saturating_add(c * a);
}

fn main() {
    println!("result: {}", calculation(120, 200, 248));
}
```

<https://doc.rust-lang.org/std/primitive.u8.html>

Type inference - Wir müssen selten Typen schreiben

```
fn use_u32(x: u32) {  
    println!("u32: {}", x);  
}  
  
fn use_i8(x: i8) {  
    println!("i8: {}", x);  
}  
  
fn main() {  
    let x = 10;  
    let y = 20;  
    use_u32(x);  
    use_i8(y);  
}
```

Typenumwandlung

`as` überprüft die zu konvertierende Zahl nicht!
Lösung: ` `.into()`

```
fn use_u32(x: u32) {  
    println!("u32: {}", x);  
}  
  
fn use_i8(x: i8) {  
    println!("i8: {}", x);  
}  
  
fn main() {  
    let y = 20;  
    use_u32(y as u32);  
    use_i8(y);  
}
```

Hands-on 11

```
/// Fibonacci sequenz
///
/// fib(0) = 0, fib(1) = 1, dann: fib(n) = fib(n-1) + fib(n-2)
fn fib(n: u32) -> u32 {
    if n < 2 {
        return todo!("Implement this");
    } else {
        return todo!("Implement this");
    }
}

fn main() {
    let n = 20;
    println!("fib({n}) = {}", fib(n));
}
```

Hands-on

```
fn main() {
    let name = "Alice";
    let age = 30;
    let height = 1.75;
    let city = "Wonderland";

    // format! can be used to format strings
    let greeting = format!("Hello, my name is {}.", name);
    printf!("{}", greeting);

    // println! prints text to the terminal
    // Experiment with different formatting options
    println!("Name: {}, Age: {}, City: {}", name, age, city);
    println!("Age: {}, Name: {}, Height: {:.2}", name, age, height);
    println!("Height: {:.2} meters, City: {}", height);

    // dbg! is for debugging
    // dbg!(greeting);
}
```

<https://doc.rust-lang.org/std/fmt/#syntax>

Blöcke und Datenfluss

„Alles“ in Rust ist ein Ausdruck

```
fn main() {  
    let ham = 13;  
    let breakfast = {  
        let eggs = 29;  
        eggs + ham  
    };  
    // dbg!(eggs);  
    dbg!(breakfast);  
}
```

Blöcke haben einen Wert, dieser kann zugewiesen werden...

...der Grund warum “const by default” funktioniert

Rust verwendet “normales” scoping

If

```
fn func(x: i32) {
    if x == 0 {
        println!("null");
    } else if x < 0 {
        println!("negative");
    } else {
        println!("positive");
    }

    let estimate = if x.abs() < 20 { "small" } else { "big" };
    println!("{} is {}", x, estimate);
}

fn main() {
    func(0);
}
```

Match

- von oben nach unten
- kein fallthrough
- muss vollständig sein
- blöcke möglich
 - ...sind ja Ausdrücke
- `_` ist der default
- `=> ()` tut nichts

```
fn func(x: i32, flag: bool) {  
    match x {  
        1 => println!("eins"),  
        10 | 11 => println!("zehn oder elf"),  
        _ => println!("anders")  
    }  
  
    let val = match flag {  
        true => 1,  
        false => 0,  
    };  
    println!("flag wert {flag} ist {val}");  
}
```

Schleifen

loop

endlos (bis break)

```
loop {
    dbg!("blub");
    if x > 20 {
        break;
    }
}
```

while

wie in C

```
while x > 20 {
    x += 1;
    dbg!(x);
}
```

```
while x > 20 {
    if x == 10 { continue }
    dbg!(x);
}
```

for

nur für iterator

```
for x in 1..5 {
    dbg!(x)
}

for elem in [2, 5, 6, 8] {
    dbg!(elem);
}
```

Funktionen

Parameter gefolgt von Typ
der letzte Ausdruck ist automatisch zurückgegeben
return kann benutzt werden

Kein: Überladen / Variable Parameterzahl

```
fn gcd(a: u32, b: u32) -> u32 {  
    if b > 0 {  
        gcd(b, a % b)  
    } else {  
        a  
    }  
}
```

Hands-on 21

```
// TODO implement functions to calculate the sum
// of all integers from 0 to 20 that are not
// divisible by 3
// - use both for and while loops
// - use both if and match

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_sum() {
        assert_eq!(sum(), 127);
    }
}
```

Tuples, Structs, and Enums

Tuple

- Gruppierte Variablen
- Oft als return-Wert benutzt
- () ist „void“

```
fn example() -> (u32, usize) {  
    (5, 10)  
}  
fn main() {  
    let ex = example();  
    println!("u32: {}", ex.0);  
    let (a, b) = example();  
    println!("a: {}, b: {}", a, b);  
}
```

<https://doc.rust-lang.org/book/ch03-02-data-types.html#compound-types>
<https://doc.rust-lang.org/std/primitive.tuple.html>

Array

- Mehrere Wege zu initialisieren
- `[1, 2]` vs `[1; 2]`
- Indizes sind gechecked
- `{ :? }` -> debug Ausgabe

```
fn example() {  
    let arr = [1, 2, 3, 4, 5];  
    let arr0 = [0; 5];  
  
    println!("{:?}", arr[3]);  
  
    for x in arr.iter() {  
        println!("{}", x);  
    }  
}
```

<https://doc.rust-lang.org/book/ch03-02-data-types.html#compound-types>
<https://doc.rust-lang.org/std/primitive.array.html>

Struct

```
pub struct Point {  
    pub x: f32,  
    pub y: f32,  
}  
  
fn example() -> Point {  
    Point { x: 42.0, y: 4.0 }  
}  
  
fn modify() -> Point {  
    let mut temp = example();  
    temp.y = 100;  
    Point { x: 100, ..temp }  
}
```

```
pub struct TupleStruct(u8, u32);  
  
fn example() -> TupleStruct {  
    TupleStruct(42, 0xDEADBEEF)  
}  
  
fn modify() -> TupleStruct {  
    let mut temp = example();  
    temp.0 = 100;  
    TupleStruct(100, temp.1)  
}
```

Enum

```
#[derive(Debug, Copy)]
pub enum Dir {
    Left = 1,
    Right,
    Up,
    Down,
}
fn example() -> Dir {
    println!("{:?}", Dir::Left);
    Dir::Left
}
```

```
#[derive(Debug, Copy)]
pub enum Cmd {
    Move(Dir),
    Jump { x: u32, y: u32 },
    Exit,
}
fn example() -> Cmd {
    println!(
        "{:?}", 
        Cmd::Move(Dir::Left)
    );
    Cmd::Jump { x: 1, y: 2 }
}
```

Methods

```
impl Point {  
    pub fn new(x: f32, y: f32) -> Self {  
        Point { x, y }  
    }  
    pub fn get_x(&self) -> f32 {  
        self.a  
    }  
    pub fn set_x(&mut self, x: f32) {  
        self.a = a;  
    }  
}
```

```
impl Point {  
    pub fn length(&self) -> f32 {  
        (self.x * self.x + self.y * self.y).sqrt()  
    }  
}
```

```
fn example() -> Point {  
    let p = Point::new(3, 4);  
    dbg!(p.length());  
    p  
}
```

Traits

```
trait Measureable {
    fn length(&self) -> f32;
}

impl Measureable for Point {
    fn length(&self) -> f32 {
        (self.x * self.x + self.y * self.y).sqrt()
    }
}

trait Animal: Measureable {
    fn sound(&self) -> String;
}
```

<https://github.com/pretzelhammer/rust-blog/blob/master/posts/tour-of-rusts-standard-library-traits.md#index--indexmut>

Generische Funktionen

```
trait Named {  
    fn name(&self) -> String;  
}  
struct Person {  
    name: String,  
}  
impl Named for Person { ... }
```

Laufzeit

```
fn print_name(named: &dyn Named)  
{  
    println!(  
        "Name: {}",  
        named.name()  
    );  
}
```

Compilieren

```
fn print_name<T: Named>(named: &T) {  
    println!(  
        "Name: {}",  
        named.name()  
    );  
}
```

Standard Traits

- Marker traits are not really implemented
 - [Copy](#): this type should be copied, not moved
 - [Send](#): type can be moved to another thread
 - [Sync](#): can be used safely from multiple threads concurrently
- “Normal traits”
 - [Clone](#): clone method to copy data
 - [Debug](#): format to string for debug output via `debug` – normal nicht manuell implementiert
 - [Eq](#) & [PartialEq](#): can be compared
 - [Ord](#) & [PartialOrd](#): can be ordered
 - [From/TryFrom](#) & [Into/TryInto](#): type conversion

Hands-on 31

```
/// Transpose a 3x3 matrix
fn transpose(matrix: [[i32; 3]; 3]) -> [[i32; 3]; 3]
```

Temporäres Array mit gleicher Dimension

Schleife die umkopiert

Temporäres Array zurückgeben

(Array wird kopiert, also in-place möglich)

Pattern matching

Match für enums

```
fn example(dir: Dir) {  
    match dir {  
        Dir::Left => println!("Left"),  
        Dir::Right => println!("Right"),  
        Dir::Up | Dir::Down => println!("Up or Down"),  
    }  
}
```

<https://doc.rust-lang.org/book/ch06-02-match.html>

<https://doc.rust-lang.org/book/ch19-03-pattern-syntax.html>

except `matches!()` <https://doc.rust-lang.org/std/macro.matches.html>

Structs / Tuples zerlegen

```
fn match_struct(p: Point) {  
    match p {  
        Point { x: 0, .. } => todo!(),  
        Point { y, .. } if y.abs() > 100.0 => todo!(),  
        p @ Point { y: 0, .. } => dbg!(p),  
        _ => todo!(),  
    }  
}
```

Comparison to C

```
typedef enum { CMD_MOVE, CMD_JUMP, CMD_EXIT } CmdType;

typedef struct {
    CmdType type;
    union {
        struct {
            Dir dir;
        } move;
        struct {
            uint32_t x;
            uint32_t y;
        } jump;
    };
} Cmd;

void debug(Cmd cmd) {
    switch (cmd.type) {
        case CMD_MOVE:
            std::cout << "Move " << cmd.move.dir << std::endl;
            break;
        case CMD_JUMP:
            std::cout << "Jump to (" << cmd.jump.x
                        << ", " << cmd.jump.y << ")" << std::endl;
            break;
        case CMD_EXIT:
            std::cout << "Exit" << std::endl;
            break;
    }
}
```

Oder abgekürzt für einen Fall

```
fn let_if(cmd: Cmd) {  
    if let Cmd::Move(dir) = cmd {  
        println!("{}:?", dir);  
    } else {  
        println!("Not a move command");  
    }  
}
```

```
fn let_destructure(cmd: Cmd) {  
    let Cmd::Jump { x, y } = cmd else {  
        return;  
    };  
    dbg!(x, y);  
}
```

<https://doc.rust-lang.org/book/ch06-03-if-let.html>

Hands-on 41

```
#[derive(Debug)]
pub enum Operation {
    Add,
    Sub,
    Mul,
    Div,
}

#[derive(Debug)]
pub enum Expression {
    Number(i32),
    Operation(Operation, Box<Expression>, Box<Expression>),
}

fn eval(e: Expression) -> i32 {
    todo!();
}
```

* dereferenziert Box

Result & Option

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

beinhaltet Wert oder Fehler

Standard für Fehlerbehandlung

```
enum Option<T> {  
    Some(T),  
    None,  
}
```

optionaler Wert

Optionale Parameter
z.B. Suchergebnis, Array Zugriff
Fehlerbehandlung

Error handling

```
fn to_u32(x: i32) -> Option<u32> {
    if x < 0 {
        None
    } else {
        Some(x as u32)
    }
}

#[test]
fn test_to_u32() {
    assert_eq!(to_u32(-1), None);
    assert_eq!(to_u32(0), Some(0));
    assert_eq!(to_u32(100).unwrap(), 100);
}
```

<https://doc.rust-lang.org/std/option/index.html>

Error handling

```
enum Error { SomeError, OtherError, }

fn fallible() -> Result<u32, Error> {
    todo!();
}

fn example() {
    let result = fallible();
    match result {
        Ok(value) => println!("Value: {}", value),
        Err(Error::SomeError) => println!("Some error"),
        Err(Error::OtherError) => println!("Other error"),
    }
}
```

<https://doc.rust-lang.org/book/ch09-00-error-handling.html>

<https://doc.rust-lang.org/std/result/index.html>

Error handling - shortcut

```
fn fallible() -> Result<u32, Error> {
    todo!()
}

fn example() -> Result<u32, Error> {
    let result = fallible()?;
    println!("Result: {:?}", result);
    Ok(())
}

fn fallible_void() -> Result<(), Error> {
    todo!()
}
```

<https://doc.rust-lang.org/book/ch09-02-recoverable-errors-with-result.html>

Hands-on 51

Hands-on 41 anpassen:

Lösung von 41 kopieren
ein **Error** enum dazubauen (dividieren durch 0)
erweitern das ein **Result<i32, Error>** returned wird
(verwende **?** für Fehler der sub-Aufrufe)

```
#[derive(Debug, Clone, Eq, PartialEq)]  
pub enum Error { ... }
```

Referenzen & Lifetimes

Referenzen

```
fn shared_ref() {  
    let a = 'A';  
    let b = 'B';  
  
    let mut r: &char = &a;  
    dbg!(*r);  
  
    r = &b;  
    dbg!(*r);  
}
```

das Ziel kann verändert werden
der Inhalt nicht

```
fn exclusive_ref() {  
    let mut point = (1, 2);  
    let x = &mut point.0;  
    // can't access point here  
    // point.1 = 3; // error  
    *x += 1;  
    println!("point: {:?}", point);  
}
```

Ziel und Inhalt sind veränderbar
“Es kann nur eine geben”

Borrowchecker - Entweder / Oder

Ein Besitzer

„Owner“

Viele konstante Referenzen

Eine veränderbare Referenz

„borrow“

„borrow“

„immutable reference“

„exclusive reference“
„mutable reference“

Verletzung -> Compilierfehler
Speicherfreigabe: Verlassen des Scopes

Borrowchecker

+

Striktes Typsystem

+

Bounds checks

=

Safe Code

Slices

```
fn example() {  
    let a = [3, 1, 4, 1, 5, 9];  
    println!("a: {a:?}");  
  
    let slice = &a[1..4];  
    println!("slice: {slice:?}");  
}
```

Teil eines Arrays

Kennt seine Größe (überprüft)
“fat pointer” (Pointer & Länge)

<https://doc.rust-lang.org/std/primitive.slice.html>

Vector

`Vec<T>` = dynamisch wachsendes Array

```
fn example() {
    let mut v1: Vec<i32> = Vec::new();
    v1.push(42);
    println!("v1: len = {}, capacity = {}", v1.len(), v1.capacity());

    if let Some(x) = v1.get(5) { println!("impossible"); }

    let mut v3 = vec![0, 0, 1, 2, 3, 4];
    v3.retain(|x| x % 2 == 0);
    println!("{}{v3:?}{}");

    v3.dedup();
    println!("{}{v3:?}{}");
}
```

<https://doc.rust-lang.org/std/vec/index.html>

Strings

String

wachsender “Container” für Zeichen

&str

ein Slice eines Strings

```
fn example() {  
    let s1: &str = "World";  
    let mut s = String::from("Hello ");  
    s.push_str(s1);  
  
    println!("s: {s}");  
  
    let r: &str = &s[0..5];  
  
    // s kann nicht modifiziert werden  
    // s.push_str("!!!!"); // error  
  
    println!("r: {r}");  
}
```

<https://doc.rust-lang.org/book/ch08-02-strings.html>

<https://doc.rust-lang.org/book/ch04-03-slices.html#string-slices>

Lifetimes

```
fn longest(x: &str, y: &str) -> &str {  
    if x.len() > y.len() { x } else { y }  
}
```

Der Compiler kann nicht wissen welche Lifetime das Resultat hat

Wir müssen explizit eine Lifetime angeben

Lifetimes

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {  
    if x.len() > y.len() { x } else { y }  
}
```

Wir sagen damit:

Es gibt eine Lifetime '*a*

x und *y* leben zumindest so lange wie '*a*

beide leben länger als das Resultat

Der Compiler überprüft innerhalb der Methode, und beim Aufruf

<https://doc.rust-lang.org/book/ch10-03-lifetime-syntax.html#lifetime-annotations-in-function-signatures>

Lifetimes - Struct

- Lifetime tracking geht weiter
 - verhindert unerwartetes modifizieren
 - gibt Daten frei

```
struct ParseResult<'a> {
    number: usize,
    animal: &'a str,
}

fn decode<'a>(input: &'a str) ->
Result<ParseResult<'a>, Error> {
    todo!()
}

fn main() {
    let mut input = "15 Bear";
    let result = decode(input);

    // input kann nicht geändert werden
    println!("result: {:?}", result);
}
```

Hands-on 61

Funktion:

3 Strings input -> längster String output

Was sind die Implikationen auf Veränderbarkeit?

Welche Einschränkungen bekommen wir?

Welche Fehler verhindert das?

Iterator

Warum

Wir wollen eigentlich nicht mittels Index iterieren

```
for idx in 0..arr.len() {  
    dbg!(arr[idx]);  
}
```

Warum?

collect

Iteratoren sind lazy, und iterieren nur wenn notwendig

```
fn example() {  
    let x = vec![1, 2, 3, 5, 8, 13];  
    let squares: Vec<i32> = x.iter().map(|x| x * x).collect();  
  
    let squares = x.iter().map(|x| x * x).collect::<Vec<_>>();  
}
```

`collect` sammelt Werte in eine neue Liste

<https://doc.rust-lang.org/book/ch13-02-iterators.html>

<https://doc.rust-lang.org/std/iter/trait.Iterator.html>

Hilfsfunktionen

70+ Hilfsfunktionen

```
fn example() {  
    let result: i32 = (1..=10) // 1 bis 10  
        .filter(|x| x % 2 == 0) // nur gerade Zahlen  
        .map(|x| x * x) // quadrieren  
        .sum(); // aufsummieren  
}
```

`sum` konsumiert, gleich wie `collect`

<https://doc.rust-lang.org/std/iter/trait.Iterator.html>

Option/Result

Monaden...

```
Some(5)
    .map(|x| x + 1)
    .and_then(|x| Some(x * 2))
    .map(|x| x + 3)
    .unwrap_or(0)
    .try_into()?;
```

Kurzversion von vielen Standard-Operationen

<https://doc.rust-lang.org/std/option/index.html>
<https://doc.rust-lang.org/std/result/index.html>

Iterator als Parser

Iteratoren tauchen an vielen Stellen auf

```
fn parse() -> Result<(usize, String), Error> {
    let mut input = "15 Bear".split(' ');

    let number: usize = input
        .next()
        .and_then(|x| x.parse().ok())
        .ok_or(Error::InvalidInput)?;

    let animal = input.next().ok_or(Error::NoName)?;

    Ok((number, animal.to_string()))
}
```

Hands-on 71

Implementieren von Hands-on 21

...mittels Iterators
`(0..20)` ist schon ein iterator

siehe:

<https://doc.rust-lang.org/std/iter/trait.Iterator.html#method.filter>
<https://doc.rust-lang.org/std/iter/trait.Iterator.html#method.sum>

Smart Pointer

Bis jetzt nur Daten am Stack
(ausser Vec<T>)

```
struct Node {  
    left: Node,  
    right: Node,  
}
```

Wie groß ist **Node**?

Bis jetzt nur Daten am Stack
(ausser Vec<T>)

```
struct Node {  
    left: Node,  
    right: Node,  
}
```

Kann das funktionieren?

Option?

```
struct Node {  
    left: Option<&Node>,  
    right: Option<&Node>,  
}
```

Noch immer am Stack...
Benötigt explizite Lifetimes
Schwer zu verwenden

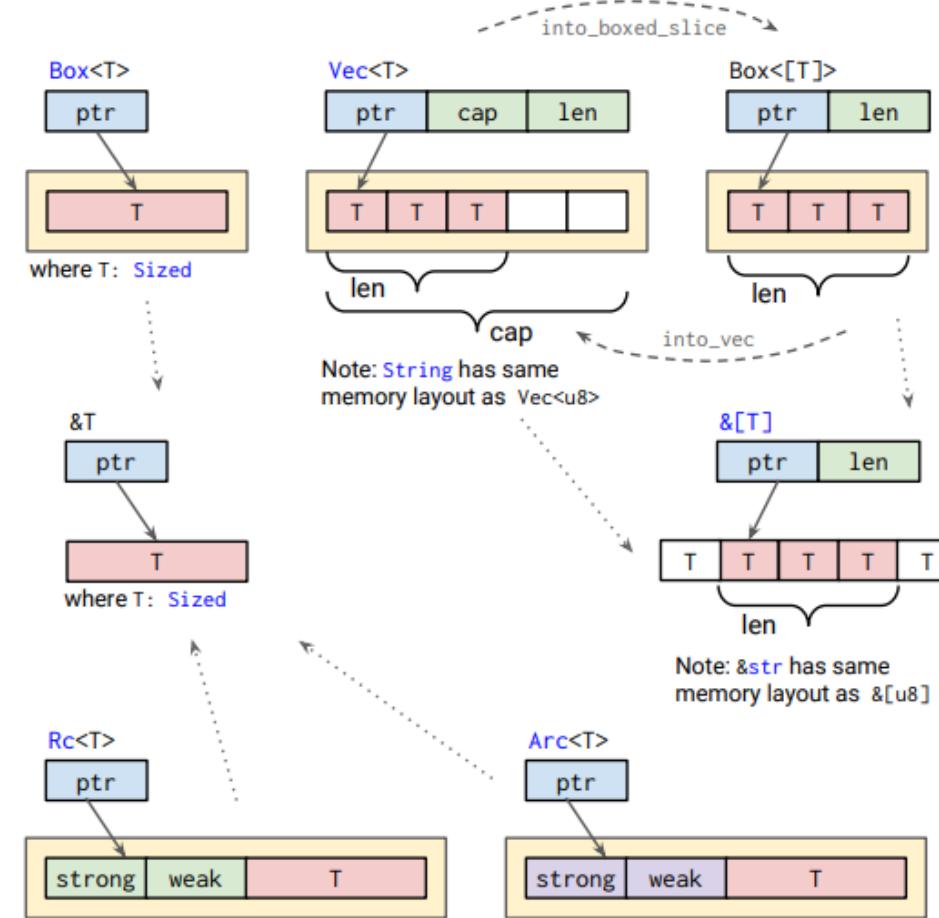
Smart Pointers

	Box<T>	Rc<T>	Arc<T>	RefCell<T>	Weak<T>
Ownership	single	multiple	multiple	single	non-owning
Threadsafe ⁽¹⁾	YES	NO	YES	NO	
Heap	YES	YES	YES	NO	NO
Ref Counted	NO	YES	YES	NO	YES
Mutable	YES	NO	NO	YES ⁽²⁾	NO
Compile time borrowchk	YES	YES	YES	NO ⁽²⁾	YES

(1) only the pointer itself, not the contained data

(2) checked at runtime

Smart Pointers



Quelle: Taesoo Kim - <https://tc.gts3.org/cs3210/2020/spring/l/lec09/lec09.html#cover>

Outlook & Recap

Safe?

Sane Standard-Bibliothek

Einfache Typen mit Features

Konsistente Fehlerbehandlung

Boundary checks

Borrow checker

Constant by-default

Pattern matching

Send/Sync Marker

leichtes Testen

Keine null-pointer

Striktes Typsystem

Rust als erste Sprache?

NEIN?

-> Python...

Statt C?

Pointer vs Referenzen...

Tooling (Buildsystem, Debugging, ...)

Macros (Debug, ...)

Gute Kultur bezgl. Dokumentation, Beispiele, etc.

Bibliotheken...

Embedded – Web – CLI – UI

Ist Rust zu kompliziert?

vs. Python?

Größte Komplexität im borrow checker
... nur ein Thema mit Referenzen

Warum nicht einfach ohne Referenzen?
.clone() überall?

<https://www.hezmatt.org/~mpalmer/blog/2024/05/01/the-mediocre-programmers-guide-to-rust.html>

Weiterführende Quellen

- Rust Cookbook: <https://rust-lang-nursery.github.io/rust-cookbook/intro.html>
- Rust by example: <https://doc.rust-lang.org/rust-by-example/>
- Comprehensive Rust: <https://google.github.io/comprehensive-rust>
- Rust Book: <https://doc.rust-lang.org/>
- <https://this-week-in-rust.org/>
- <https://www.theembeddedrustacean.com/>
- Crust of Rust: <https://www.youtube.com/playlist?list=PLqbS7AVVErFiWDOAVrPt7aYmnuuOLYvOa>
- The Rusty Bits (embedded): <https://www.youtube.com/@therustybits>

**Feedback
bitte**

-> E-Mail





- [https://www.ph-online.ac.at/phst/ee/ui/ca2/app/desktop/#/slc.tm.cp/student/courses/498947?ctx=design=ca&\\$scrollTo=toc_overview](https://www.ph-online.ac.at/phst/ee/ui/ca2/app/desktop/#/slc.tm.cp/student/courses/498947?ctx=design=ca&$scrollTo=toc_overview)