AAU SATLAB

RUST INTRODUCTION

By Albert Laursen

CONTENTS

- What is Rust
- Why Choose Rust
- C++/C vs Rust
- Installation
- Tooling
- Compiling
- Getting Started

WHAT IS RUST?

- Systems Programming Language

WHAT IS RUST?

- Systems Programming Language
- Compiled

WHAT IS RUST?

- Systems Programming Language
- Compiled
- Modern Language
  - Type system, concurrency, memory safety

WHAT IS RUST?

- Systems Programming Language
- Compiled
- Modern Language
  - Type system, concurrency, memory safety
- No Runtime Garbage Collector
  - Scope & Lifetimes

WHY CHOOSE RUST?

- Safety
  - No null pointers
  - Borrowing & Lifetimes

WHY CHOOSE RUST?

- Safety
  - No null pointers
  - Borrowing & Lifetimes
- Modern Tooling
  - Cargo & Crates

WHY CHOOSE RUST?

- Safety
  - No null pointers
  - Borrowing & Lifetimes
- Modern Tooling
  - Cargo & Crates
- Only modern choice for embedded systems

WHY CHOOSE RUST?

- Safety
  - No null pointers
  - Borrowing & Lifetimes
- Modern Tooling
  - Cargo & Crates
- Only modern choice for embedded systems
- Sizeable embedded community

AAU SATLAB

C++/C VS RUST

- C++/C is not safe
  - C++ is a superset of C

AAU SATLAB

C++/C VS RUST

- C++/C is not safe
  - C++ is a superset of C
- Borrow Checker
  - Compile time safety

AAU SATLAB

C++/C VS RUST

- C++/C is not safe
  - C++ is a superset of C
- Borrow Checker
  - Compile time safety
- Interfaces
  - Abstract classes vs Traits
  - e.g. `embedded-hal`

AAU SATLAB

C++/C VS RUST

- C++/C is not safe
  - C++ is a superset of C
- Borrow Checker
  - Compile time safety
- Interfaces
  - Abstract classes vs Traits
  - e.g. `embedded-hal`
- OOP
  - C++: `class`'es
  - Rust: `structs`, `enums` & `impls`

FURTHER READING

- Rust Book
  - Online Version: doc.rust-lang.org/book
  - Interactive Version: rust-book.cs.brown.edu
- Rustlings Exercises
  - Interactive Lessons: rust-lang/rustlings
- Cheatsheet
  - Cheat Sheet: cheats.rs

AAU SATLAB

INSTALL PARTY

AAU SATLAB

INSTALLATION

Go to rustup.rs and follow the instructions

Or use the following command in your terminal:

Windows:

```
winget install rustup
```

Linux/Mac:

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs |
```

TOOLING

- `rustc` - Rust Compiler
- `cargo` - Package Manager
- `rustup` - Rust Version Manager
- `rustfmt` - Code Formatter
- `clippy` - Linter

COMPILING

```
cargo new hello_world
cd hello_world
cargo build
cargo run
```

```
Hello, world!
```

VISUAL STUDIO CODE

Extension	Description	Rec
rust-analtzer	Language Server, Formatter, Linter	
Even Better TOML	TOML language support, primarily for cargo	
CodeLLDB	Debugger for Host applications	
Dependi	Dependency Tool for cargo config	
Error Lens	Better Error Highlightning	

CODE EXAMPLES

Get git repository:

```
git clone https://github.com/Awlaursen/rust-examples.git
```

BASIC SYNTAX

File: `01-hello\src\main.rs`

```rust
// Primitive data types in Rust
let x: i32 = 5; // signed 32-bit integer
let y: f64 = 2.5; // 64-bit floating point
let z: u32 = 1_000_000; // unsigned 32-bit integer
let a: char = 'a'; // single Unicode character
let b: bool = true; // boolean
let c: &str = "Hello, world!"; // string slice
```

BASIC SYNTAX

File: `01-hello\src\main.rs`

```
1 // inferred data types in Rust
2 let x = 5; // i32
3 let y = 2.5; // f64
4 let z = 1_000_000; // i32
```

```
1 // Explicitly specifying data types in Rust
2 let x = 5i32;
3 let y = 2.5f64;
4 let z = 1_000_000u32;
```

BASIC SYNTAX

File: `01-hello\src\main.rs`

```rust
// Compound data types in Rust
let d = [1, 2, 3, 4, 5]; // [i32; 5]
let e = (1, 2, 3, 4.5, 5.5); // (i32, i32, i32, f64, f64)
```

```rust
// Accessing elements in compound data types
let first = d[0];
let second = e.1;
print!("first = {}, second = {}", first, second);
```

BASIC SYNTAX

File: `01-hello\src\main.rs`

```rust
// Control flow in Rust
if x < 5 {
    println!("x is less than 5");
} else if x == 5 {
    println!("x is equal to 5");
} else {
    println!("x is greater than 5");
}
```

BASIC SYNTAX

File: `01-hello\src\main.rs`

```rust
// Loops in Rust
for i in 0..5 {
    println!("i = {}", i);
}

let mut i = 0;
while i < 5 {
    println!("i = {}", i);
    i += 1;
}
```

```rust
// Loop forever until break
let mut i = 0;
loop {
    println!("i = {}", i);
    i += 1;
    if i == 5 {
        break;
    }
}
```

BASIC SYNTAX

File: `01-hello\src\main.rs`

```rust
// Mutability in Rust
let mut x = 5;
x = 10;
```

```rust
// Shadowing in Rust
let x = 5;
let x = x + 1;
let x = x * 2;
```

AAU SATLAB

BASIC SYNTAX

File: 01-hello\src\main.rs

```rust
// Macros in Rust
let my_vector = vec![1, 2, 3, 4, 5]; // Vec<i32>
println!("x = {}, y = {}, z = {}", x, y, z);

println!("{:?}", my_vector);
```

```rust
let my_vector = <[_]>::into_vec(
    #[rustc_box]
    alloc::boxed::Box::new([1, 2, 3, 4, 5]),
); // Vec<i32>

std::io::_print(
  builtin #format_args("x = {}, y = {}, z = {}\n",x,y,z));
```

FUNCTIONS

File: `02-functions\src\main.rs`

```rust
// Function with no return value
fn print_result(result: i32) {
    println!("result is: {}", result);
}

// Being explicit about return value
fn print_result(result: i32) -> () {
    println!("result is: {}", result);
    return (); // Or just ()
}
```

FUNCTIONS

File: `02-functions\src\main.rs`

```rust
// Function with return value
fn add_numbers(x: i32, y: i32) -> i32 {
    return x + y;
}

// Function with multiple return values
fn add_and_multiply(x: i32, y: i32) -> (i32, i32) {
    (x + y, x * y)
}
```

AAU SATLAB

OWNERSHIP

File: `03-ownership\src\bin\01-ownership.rs`

```
1 // Scopes are defined by blocks like this one or functions
2 {                           // s is not valid here, not yet decl
3     let s = "hello";        // s is valid from this point forwar
4
5     // do stuff with s
6     println!("{}", s);
7 }                           // this scope is now over, and s is
8                             // longer valid
9 println!("{}", s); // This will not compile
```

OWNERSHIP

```
error[E0425]: cannot find value `s` in this scope
  --> src\bin\01-ownership.rs:10:20
   |
 9 |     println!("{}", s);
   |                    ^ not found in this scope

For more information about this error, try `rustc --explain E0
error: could not compile `structs` (bin "01-ownership") due to
       previous error
```

AAU SATLAB

OWNERSHIP

File: `03-ownership\src\bin\01-ownership.rs`

```rust
// Move of ownership from s1 to s2
let s1 = String::from("hello");
let s2 = s1;  // Move happens because String is a heap-alloc
              // type and it is not a `Copy` type

println!("{s1}, world!"); // This will not compile
```

AAU SATLAB

OWNERSHIP

```
error[E0382]: borrow of moved value: `s1`
  --> src\bin\01-ownership.rs:17:15
   |
13 |     let s1 = String::from("hello");
   |         -- move occurs because `s1` has type `String`, wh
                              does not implement the `Copy` tr
14 |     let s2 = s1;
   |              -- value moved here
...
17 |     println!("{s1}, world!");
   |              ^^^^ value borrowed here after move
   |
```

```
help: consider cloning the value if the performance cost is ok
   |
14 |     let s2 = s1.clone();
   |               ++++++++
```

OWNERSHIP

File: `03-ownership\src\bin\01-ownership.rs`

```
1 // Clone of s3 to s4
2     let s3 = String::from("hello");
3     let s4 = s3.clone();    // Clone is called explicitly to
4
5     println!("{}, world!", s3); // This will compile
```

AAU SATLAB

OWNERSHIP

File: `03-ownership\src\bin\01-ownership.rs`

```rust
// Ownership and functions
let s = String::from("hello"); // s comes into scope

takes_ownership(s);   // s's value moves into the function.
                      // ... and so is no longer valid here

let x = 5;            // x comes into scope

makes_copy(x);        // x would move into the function,
                      // but i32 is Copy, so it's okay stil
                      // use x afterward
```

OWNERSHIP

File: `03-ownership\src\bin\01-ownership.rs`

```rust
fn takes_ownership(s: String) { // s comes into scope
    println!("{}", s);
} // Here, s goes out of scope and `drop` is called.

fn makes_copy(i: i32) { // i comes into scope
    println!("{}", i);
} // Here, i goes out of scope. Nothing special happens.
```

AAU SATLAB

OWNERSHIP

File: `03-ownership\src\bin\01-ownership.rs`

```rust
let s2 = String::from("hello");     // s2 comes into scope

let s3 = takes_gives_back(s2);  // s2 is moved into
                                // takes_gives_back, which
                                // moves its return value i

// This function takes a String and returns one
fn takes_gives_back(s: String) -> String {  // s comes into
  println!("{}", s);
  s  // s is returned and moves out to the calling function
}
```

AAU SATLAB

BORROWING

File: `03-ownership\src\bin\02-borrowing.rs`

Syntax	Description
`&T`	immutable reference
`&mut T`	mutable reference
`&str`	string slice
`&[T]`	slice

AAU SATLAB

BORROWING

File: `03-ownership\src\bin\02-borrowing.rs`

```rust
let s1 = String::from("hello");

// Function takes an immutable reference
let len = calculate_length(&s1);

println!("The length of '{s1}' is {len}.");

fn calculate_length(s: &String) -> usize {
    // reference is {read-only / immutable / borrowed}
    s.len() // returns the length of the string without
            // needing ownership
}
```

AAU SATLAB

BORROWING

File: `03-ownership\src\bin\02-borrowing.rs`

```rust
// This block will not compile
let s = String::from("hello");

// Function takes an immutable reference
// but tries to mutate the borrowed data
change(&s);

fn change(some_string: &String) {
    // mutability is not allowed for borrowed references
    some_string.push_str(", world"); // This will not compi
}
```

BORROWING

File: `03-ownership\src\bin\02-borrowing.rs`

```rust
// This block will not compile
let mut s = String::from("hello");

// There can be only one mutable reference to a
// particular piece of data at a time
let r1 = &mut s;
let r2 = &mut s;

println!("{}, {}", r1, r2);
```

BORROWING

File: `03-ownership\src\bin\02-borrowing.rs`

```
1 // This block will compile
2 let mut s = String::from("hello");
3
4 {
5     let r1 = &mut s;
6 } // r1 goes out of scope here, so we can make a new
7   // reference with no problems.
8
9 let r2 = &mut s;
```

AAU SATLAB

BORROWING

File: `03-ownership\src\bin\02-borrowing.rs`

```rust
// This block will compile
let mut s = String::from("hello");

let r1 = &s; // no problem
let r2 = &s; // no problem
println!("{r1} and {r2}");
// variables r1 and r2 will not be used after this point

let r3 = &mut s; // no problem
println!("{r3}");

// Will not compile if we add the following line
// println!("{r1} and {r2}");
```

STRUCTS

File: `04-structs\src\main.rs`

```rust
struct MyStruct {};

struct Person {
    first_name: String,
    last_name: String,
    age: u8,
}

struct Point2d (i32, i32);
```

STRUCTS

File: `04-structs\src\main.rs`

```rust
// Instantiating a struct
let person = Person {
    first_name: String::from("John"),
    last_name: String::from("Doe"),
    age: 30,
};

let point = Point2d (0, 0);
```

AAU SATLAB

STRUCTS

File: `04-structs\src\main.rs`

```rust
// Access fields of the Person
println!("first_name = {}", person.first_name);
println!("last_name = {}", person.last_name);
println!("age = {}", person.age);

// Access fields of the Point2d
println!("x = {}", point.0);
println!("y = {}", point.1);
```

```rust
// or like this
let Point2d (x, y) = point;
println!("x = {}, y = {}", x, y);
```

AAU SATLAB

STRUCTS

File: `04-structs\src\main.rs`

```rust
// Structs can have associated functions & methods
impl Person {
    // associated function
    fn new(first_name: &str, last_name: &str, age: u8) -> Self {
        Person {
            first_name: first_name.to_string(),
            last_name: last_name.to_string(),
            age,
        }
    }

    // method
    fn full_name(&self) -> String {
        format!("{} {}", self.first_name, self.last_name)
    }
}
```

STRUCTS

File: `04-structs\src\main.rs`

```
1 // Using associated functions & methods
2 let person = Person::new("Jane", "Doe", 25);
3 println!("full_name = {}", person.full_name());
```

THAT'S ALL

for today...