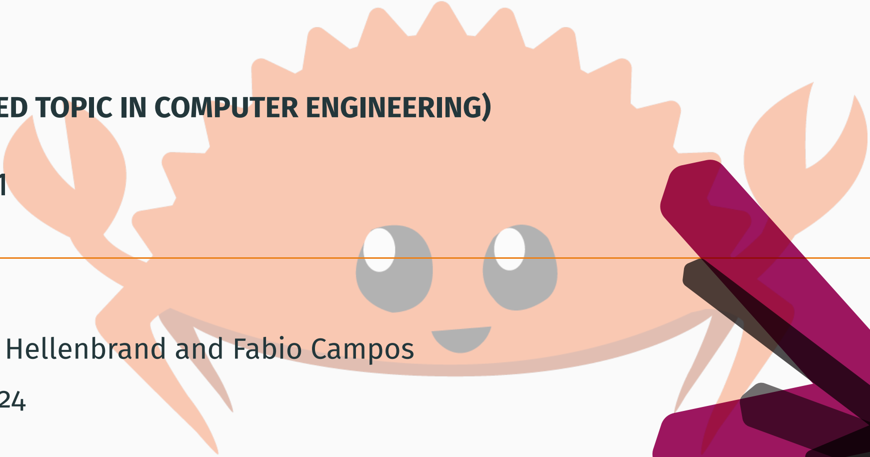
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

# Rust

## (SELECTED TOPIC IN COMPUTER ENGINEERING)

LV 7281



Andreas Hellenbrand and Fabio Campos

18.04.2024

# Agenda

- About Rust
- Variable bindings
- Primitives
- Control Flow

# About Rust

# Rust

*“Rust is a **multi-paradigm**, **general-purpose** programming language designed for **performance** and **safety**.”*[1]

[1] https://www.mongodb.com/developer/languages/rust/

# Why Rust?

### Reasons for Rust

- changes the way of thinking
- excellent documentation
- highly user-friendly compiler
- performance
- simple and safe concurrency
- strong memory safety guarantees
- open source
- growing and friendly community

# Why not Rust?

**Reasons against Rust**

- immature language
- steep learning curve $\rightarrow$ think differently
- not widely used in industry (yet)
- compiler not available for every hardware

# Timeline 1/2

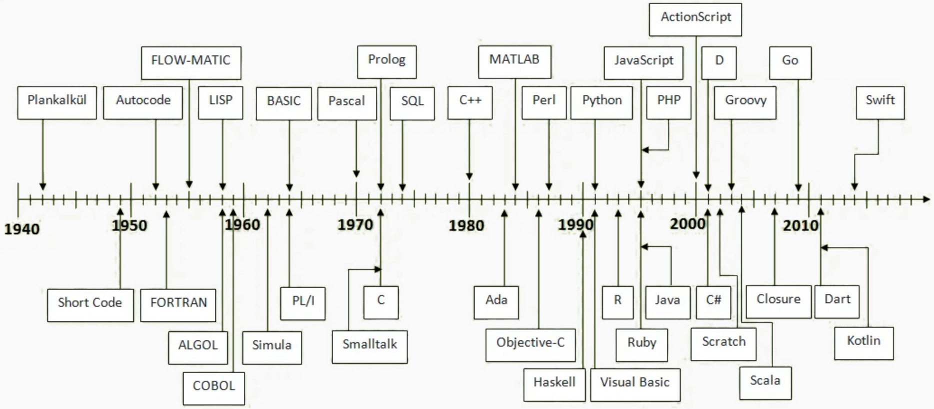


**Figure 1:** Timeline of programming languages[2]

[2] from `https://media.bevopr.io/en/io-trends-in-today-and-tomorrow-s-programming-languages`

# Timeline

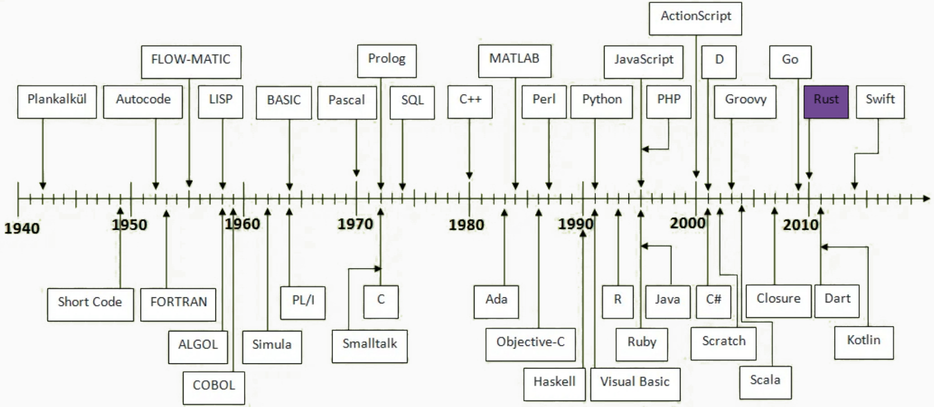
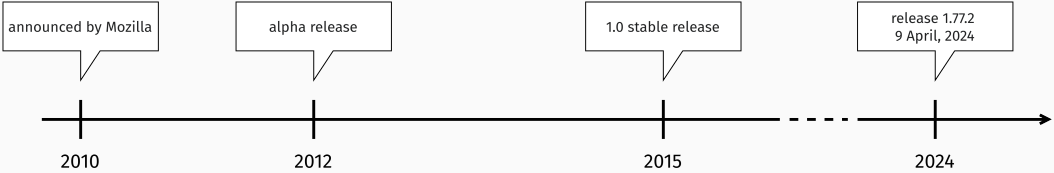


**Figure 1:** Timeline of programming languages[2]

[2] from `https://media.bevopr.io/en/io-trends-in-today-and-tomorrow-s-programming-languages`

# Timeline 2/2



announced by Mozilla

alpha release

1.0 stable release

release 1.77.2
9 April, 2024

2010

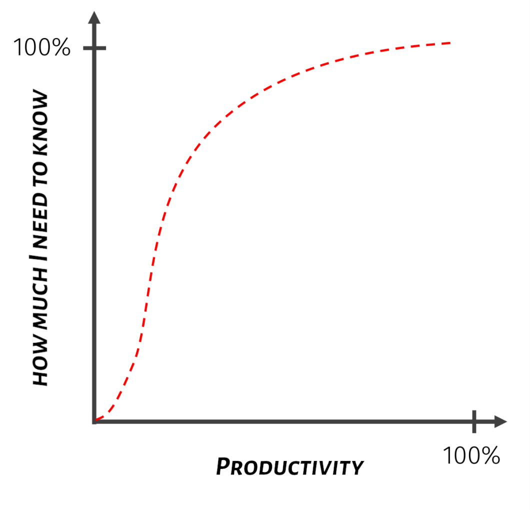
2012

2015

2024

Rust’s timeline

# Rust’s learning curve



# Language properties

- Multi-paradigm language
- Typing: static, strong, inferred
- Cross-platform: Linux, Windows, OS X, Android, ...
- Open source (community project)

# Hello World

- `fn` function declaration
- `main` starting point
- `println!` prints to the standard output, with a newline
- `!` means macro (similar to function)

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

# Variable bindings

# Mutability

- declaration: `let name: type = expr;`
- variables are **immutable by default**
- mutable by adding keyword `mut`

```rust
fn main() {
    let x = 5; // no type -> type inference
    x = 6;  // fine ?
    let mut y = 7;
    y = 8;
}
```

# Constants

**const:** an unchangeable value, explicitly typed

`const CONST_NAME: dataType = value;`

**static:** `'static` lifetime, possibly mutable variable

`static (mut) CONST_NAME: dataType = value;`

mutable → unsafe code

```rust
const THRESHOLD: i32 = 10;
static mut VALUE: i32 = 101;

fn main() {
    println!("Threshold = {}", THRESHOLD);
    println!("VALUE = {}", VALUE); // fine ?
}
```

# Shadowing & scope

allows to declare new variable with the same name as previous variable

```rust
fn main() {
    let x = 0;
    {
        let x = 1;
        println!("x = {}", x);
    }
    println!("x = {}", x);
    let x = "HSRM";
    println!("x = {}", x);

    // changing the type
    let spaces = "   ";
    let spaces = spaces.len();
}
```

# Late initialization

declare variable bindings first, and initialize them later → may lead to the use of uninitialized variables

```rust
fn main() {
    let a;
    let x = 1;
    {
        let x = 2;
        a = x * x;
    }
    println!("x = {}, a = {}", x, a);
}
```

# Primitives

# Scalar types

| | |
|---|---|
| **signed integers:** | `i8`, `i16`, `i32`, `i64` and `isize`<br>$[-(2^{n-1}), 2^{n-1} - 1]$ for $n \in \{8, 16, 32, 64\}$ |
| **unsigned integers:** | `u8`, `u16`, `u32`, `u64` and `usize`<br>$[0, 2^{n} - 1]$ for $n \in \{8, 16, 32, 64\}$ |
| **floating point:** | `f32`, `f64` → IEEE-754 standard |
| **characters:** | `char` → unicode scalar values (4 bytes) |
| **string slice:** | `str` |
| **boolean:** | `bool` [true or false] |
| **unit type:** | `()` → tuple w/o values |

# Scalar types - examples

```rust
fn main() {
    //  type inference
    let x = 2.0; // f64

    // explicit type annotation
    let y: f32 = 3.0; // f32
    let a: bool = true;
    let b: char = ' '; // unicode
    let c: i32 = 3;
    let d: f64 = 3.14;

    // type casting
    let e = y as i32;
    println!("e = {}", e);
    let f = e as bool; // fine?
}
```

# Compound types 1/2

**tuple:** variety of types, fixed length

```rust
fn main() {
    // tuple
    let t: (char, bool) = (' ', true);
    let (x, y): (char, bool) = (' ', true);
    let tuple_of_tuples = ((' ', true), (500, -1), 2024);
    // destructuring ...
    let (u, v) = t;
    let _ = t.0 == x;
    let _ = t.1 == y;
    println!("tuple_of_tuples.1.0 = {}", tuple_of_tuples.1.0);
}
```

# Compound types 1/2

**array:** same type, fixed length

```rust
fn main() {
    // type signature optional
    let numbers: [i32; 5] = [1, 2, 3, 4, 5];
    let othernumbers = [6,7,8];
    // index starts at 0
    println!("First element of the array: {}", numbers[0]);
    // all elements init to same value
    let sieve = [true; 500];
}
```

# Literals and Operators

Numeric literals can be type annotated by adding the type as a suffix

Hexadecimal, octal or binary expressed using `0x`, `0o` or `0b` notation

Underscores to improve readability

```rust
fn main() {
    // Integer addition
    println!("1 + 2 = {}", 1u32 + 2);

    // Integer subtraction
    println!("1 - 2 = {}", 1i32 - 2);
    println!("1 - 2 = {}", 1u32 - 2); // fine?

    // Bitwise operations
    println!("0011 AND 0101 is {:04b}", 0b0011u32 & 0b0101);
    println!("0011 OR 0101 is {:04b}", 0b0011u32 | 0b0101);
    println!("0011 XOR 0101 is {:04b}", 0b0011u32 ^ 0b0101);
    println!("1 << 5 is {}", 1u32 << 5);
    println!("0x80 >> 2 is 0x{:x}", 0x80u32 >> 2);

    // Use underscores to improve readability!
    println!("One million is written as {}", 1_000_000u32);
}
```

# Control Flow

# If expression

branch the code depending on conditions

```
fn main() {
    let a = 7;
    if a == 4 {
        // if branch
    } else if a > 10 {
        // else if branch
    } else {
        // else branch
    }
    if a { // fine ?
        println!("a != 0");
    }
}
```

# Loop

**loop:** repeat block until you explicitly tell it to stop

**break, continue:** apply to innermost loop at that point or to loop label

```rust
fn main() {
    let mut count = 0;
    'outer: loop {
        let mut remaining = 3;
        loop {
            println!("remaining = {remaining}");
            if remaining == 0 {
                break;
            }
            if count == 2 {
                break 'outer;
            }
            remaining -= 1;
        }
        count += 1;
    }
    println!("count = {count}");
}
```

# While

repeat block as long as condition is `true`

```rust
fn main() {
    let mut number = 3;
    while number != 0 {
        println!("number = {}", number);
        number -= 1;
    }
    println!("lift off!!!");
}
```

# While `true` vs Loop

## Equivalent?

```rust
fn main() {
    let x;
    loop { x = 1; break; }
    println!("{}", x);
}
```

```rust
fn main() {
    let x;
    while true { x = 1; break; }
    println!("{}", x);
}
```

# Functions

`fn function_name(var_name: type, ...) -> return_type {}`

```rust
fn main() {
    println!("x + y = {}", add(5, 7));
}

fn add(x: i32, y: i32) -> i32 {
    return x + y;
}

fn sub(x: i23, y: i32) -> i32 {
    x - y
}
```

# Everything is an Expression

```
fn main(){
    let x = 2;
    let y = if 25 * x > 100 {
            "BIGGER"
        } else {
            "smaller"
        };
}
```