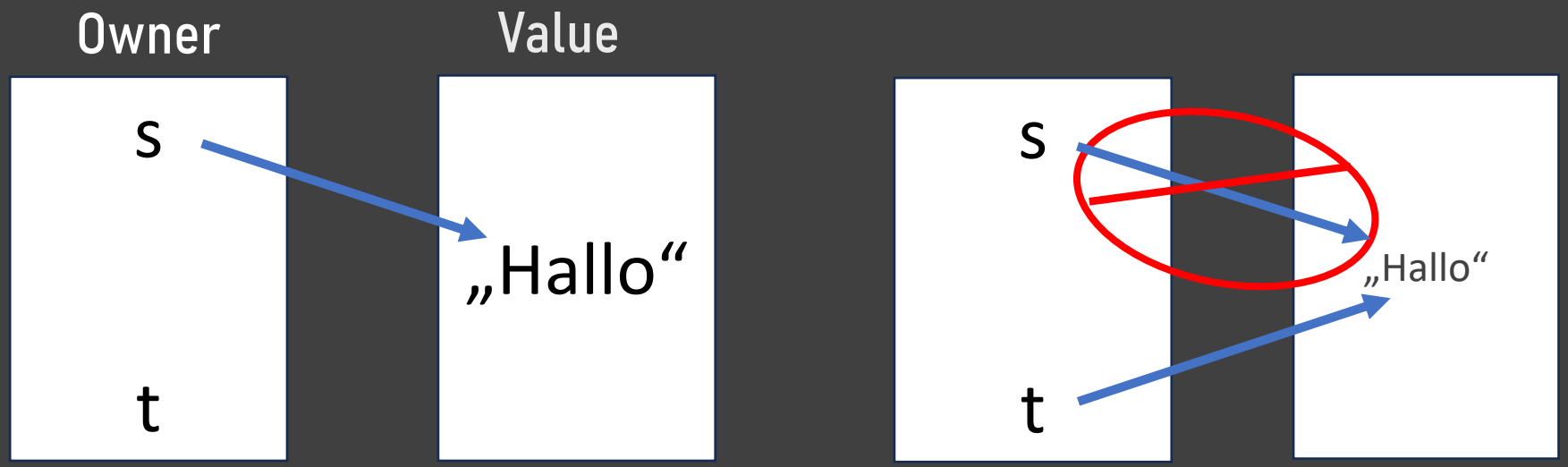
Einführung in Rust

Inhaltsverzeichnis

1. Demo

2. Besonderes an Rust

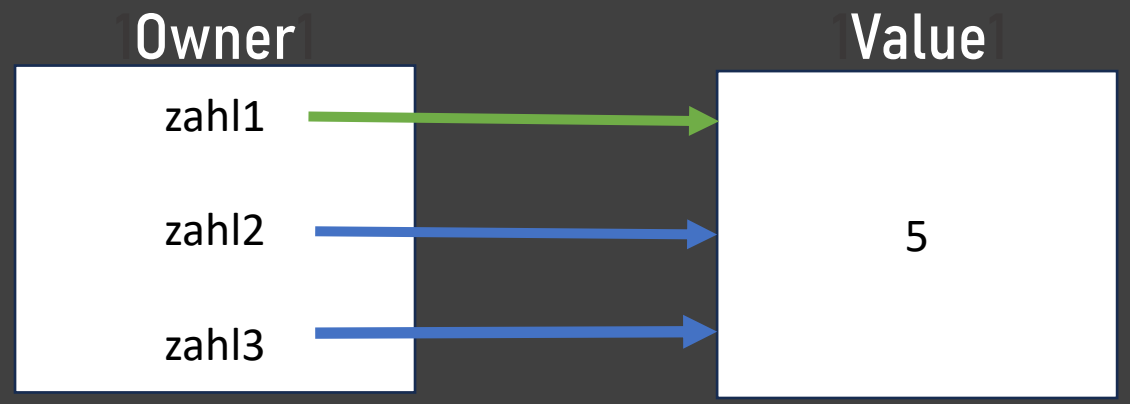
2.2. Memory Management

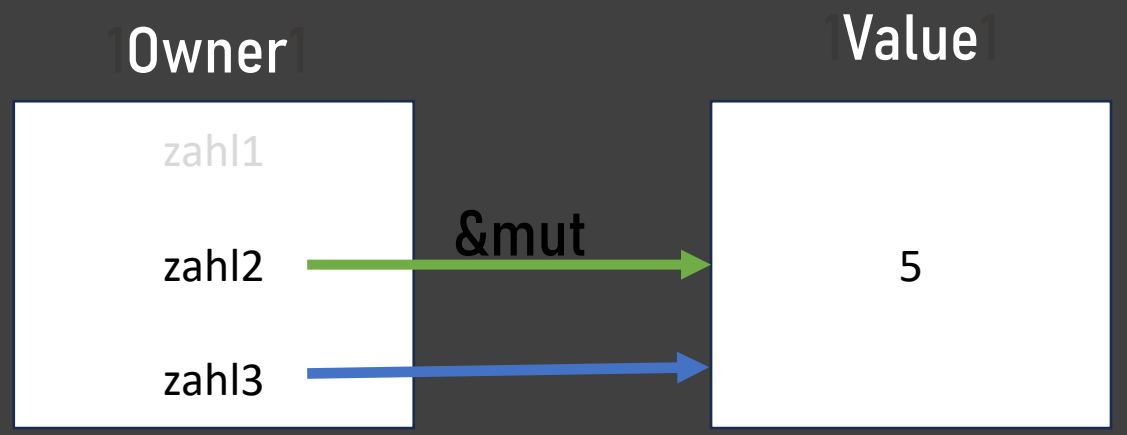


Codebeispiel:

```
let s: String = String::from("Hallo");
let t: String = s;
println!("{}", s);
```

2.3. Mutability und Borrwing





Sichert Parallelität

2.3. Fehlermanagement

Option<T>

```
Tabnine | Edit | Test | Explain | Document
fn divide(a: i32, b: i32) -> Option<i32> {
    if b == 0 {
        None
    } else {
        Some(a / b)
    }
}

Tabnine | Edit | Test | Explain | Document | ▶ Run | ⚙ Debug
fn main() {
    let result: Option<i32> = divide(a: 10, b: 2);
    match result {
        Some(value: i32) => println!("Ergebnis: {}", value),
        None => println!("Fehler: Division durch Null!"),
    }
}
```

Result<T, E>

```
use std::fs::File;
use std::io::{self, Read};

Tabnine | Edit | Test | Explain | Document
fn read_file() -> Result<String, io::Error> {
    let mut file: File = File::open(path: "test.txt")?;
    let mut content: String = String::new();
    file.read_to_string(buf: &mut content)?;
    Ok(content)
}

Tabnine | Edit | Test | Explain | Document | ▶ Run | ⚙ Debug
fn main() {
    match read_file() {
        Ok(content: String) => println!("Content: {}", content),
        Err(error: Error) => println!("Error: {}", error),
    }
}
```

Typ:	Verwendung:
Option<T>	Wert überprüfen (da, fehlt)
Result<T, E>	Kann fehlschlagen

2.4. Pattern Matching

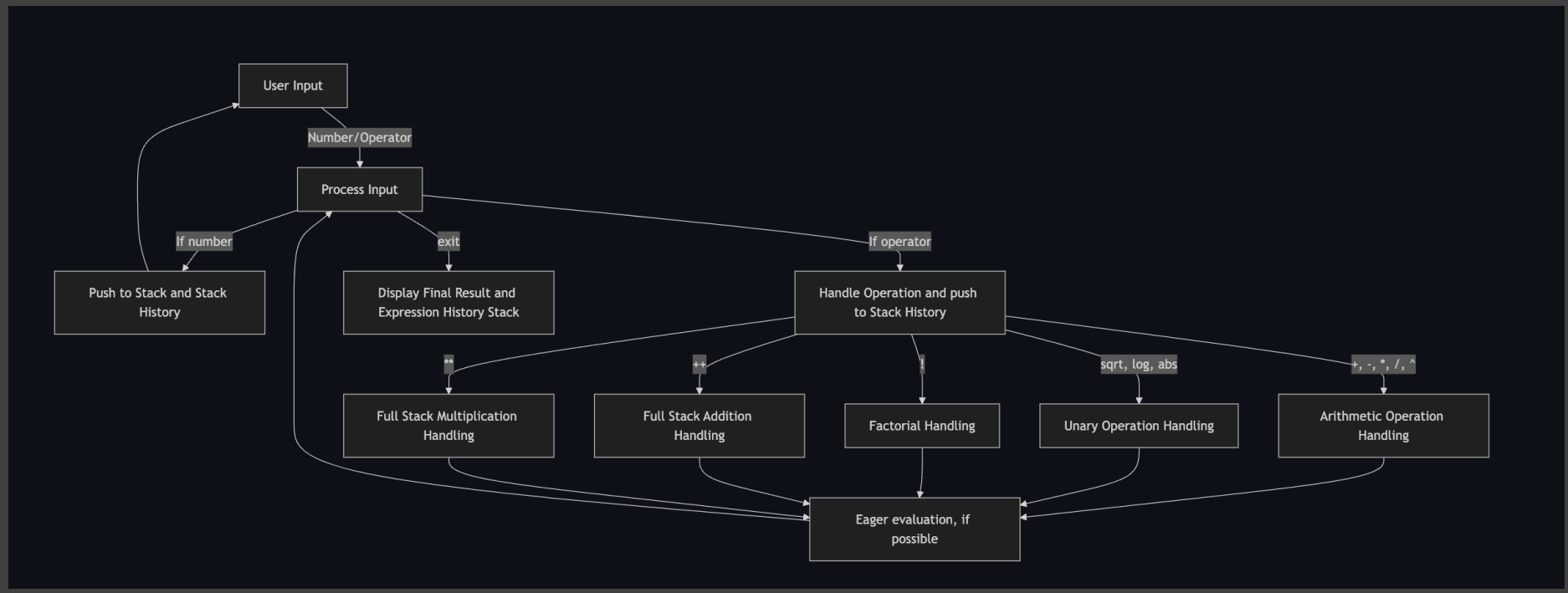
Typsicherheit und Nullsicherheit

```
Tabnine | Edit | Test | Explain | Document
fn check_number(num: Option<i32>) {
    match num {
        Some(n: i32) if n > 10 => println!("Größer als 10: {}", n),
        Some(n: i32) => println!("Kleiner als 10: {}", n),
        None => println!("Keine Zahl"),
    }
}
```

3. Rust anhand des RPN

Funktionalität

1. Demo
2. Besonderes an Rust
2.1. Memory Management
2.2. Mutability und Borrowing
2.3. Fehlermanagement
2.4. Pattern Matching
3. Rust erklärt anhand des RPN Calculators



1. Demo
2. Besonderes an Rust
2.1. Memory Management
2.2. Mutability und Borrowing
2.3. Fehlermanagement
2.4. Pattern Matching
3. Rust erklärt anhand des RPN Calculators

```rust
// Library for handling user inputs
use std::io;

// Class declaration, clone trait
#[derive(Clone)]
2 implementations
struct RPNCalculator {
    stack: Vec<f64>,
    history_stack: Vec<String>,
}

impl RPNCalculator {
    // Constructor, initializing the vectors
    fn new() -> Self {
        Self {
            stack: Vec::new(),
            history_stack: Vec::new(),
        }
    }

```

Import IO dependency

Clone Trait

Struct

Attributes

Vector data structure

Implementation block for methods

Constructor

New Vector

Changing attributes of the „Self“ object

1. Demo
2. Besonderes an Rust
2.1. Memory Management
2.2. Mutability und Borrowing
2.3. Fehlermanagement
2.4. Pattern Matching
3. Rust erklärt anhand des RPN Calculators

```rust
// Main function for executing the RPN calculator
▶ Run | ⚙ Debug
fn main() {
    let mut calc: RPNCalculator = RPNCalculator::new();
    RPNCalculator::welcome_prompt();
    let mut input: String = String::new();

    // "Main loop", repeating logic for each input
    loop {
        input.clear();
        io::stdin().read_line(buf: &mut input).unwrap();
        let input: &str = input.trim();

        if input.eq_ignore_ascii_case("exit") {
            println!("Exiting RPN Calculator...");
            println!("Your infix calculation is: {}", calc.clone().reconstruct_expression_infix());
            println!("Your LaTeX calculation is: {}", calc.clone().reconstruct_expression_latex());

            match calc.get_result() {
                Some(value: f64) => println!("The final result is: {}", value),
                None => println!("No result available."),
            }
            break;
        }

        calc.apply_operation(token: input);

        match input {
            "+" | "-" | "*" | "/" | "^" | "sqrt" | "log" | "abs" | "++" | "**" | "!" => {
                if let Some(value: f64) = calc.get_result() {
                    println!("The current result is: {}", value);
                }
            }
            _ => {}
        }
    }
} fn main
```

Initialization variable
Method calls on classes

Loop

io
unwrap()

Std function calls

Default case

1. Demo
2. Besonderes an Rust
2.1. Memory Management
2.2. Mutability und Borrowing
2.3. Fehlermanagement
2.4. Pattern Matching
3. Rust erklärt anhand des RPN Calculators

```rust
// Applying operation to the stack
fn apply_operation(&mut self, token: &str) {
    self.history_stack.push(token.to_owned());
    match token {
        "+" | "-" | "*" | "/" | "^" => {
            self.arithmetical_operation_handling(token);
        }
        "sqrt" | "log" | "abs" => {
            self.log_abs_sqrt_operation_handling(token);
        }
        "!" => {
            self.factorial_operation_handling();
        }
        "++" => {
            self.full_stack_addition_handling();
        }
        "**" => {
            self.full_stack_multiplication_handling();
        }
        _ => {
            match token.parse::<f64>() {
                Ok(_) => self.new_number_handling(token),
                Err(_) => {
                    println!("Invalid input '{}'", token);
                    self.history_stack.pop();
                }
            }
        }
    }
} fn apply_operation
```

Method call on object
push()

1. Demo
2. Besonderes an Rust
2.1. Memory Management
2.2. Mutability und Borrowing
2.3. Fehlermanagement
2.4. Pattern Matching
3. Rust erklärt anhand des RPN Calculators

```
50
51      fn arithmetical_operation_handling(&mut self, token: &str) {
52          let b: f64 = self.stack.pop().unwrap();
53          let a: f64 = self.stack.pop().unwrap();
54          let result: f64 = match token {
55              "+" => a + b,
56              "-" => a - b,
57              "*" => a * b,
58              "/" => a / b,
59              "^" => a.powf(b),
60              _ => unreachable!(),
61          };
62          self.stack.push(result);
63      }
```

1. Demo
2. Besonderes an Rust
2.1. Memory Management
2.2. Mutability und Borrowing
2.3. Fehlermanagement
2.4. Pattern Matching
3. Rust erklärt anhand des RPN Calculators

```
120     fn reconstruct_expression_infix(&mut self) -> String {
121         if let Some(token: String) = self.history_stack.pop() {
122             match token.as_str() {
123                 "+" | "-" | "*" | "/" | "^" => {
124                     let right: String = self.reconstruct_expression_infix();
125                     let left: String = self.reconstruct_expression_infix();
126                     format!("({} {} {})", left, token, right)
127                 }
```

if let Some(T) = überprüft ob Rückgabewert nicht None ist