TUM
BLOCKCHAIN
CLUB

Rust Language Portrait 

RUST BOOTCAMP
WINTER 24

Plan

00	**Overview** - *Rust in big picture*
01	**Rust Feature** - *Why is Rust cool*
02	**Basic Syntax** - *Hello world!*
03	**Collections** - *Vectors, HashMaps and other...*
04	**Project Structure** - *What to put where*
05	**Hands-On** - *Let's code BST! ~30 mins*

00 Overview

RUST IN BIG PICTURE

“For the eighth year in a row, Rust has topped the chart as the most desired programming language in Stack Overflow’s annual developer survey“

Github Blog - Why Rust is the most admired language among developers

But why Rust?

...literally copied from Rust website

01

PERFORMANCE

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

02

RELIABILITY

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

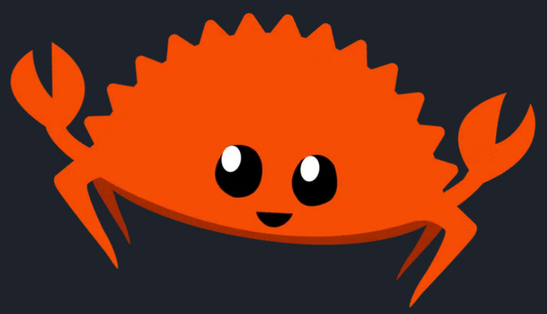
03

PRODUCTIVITY

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Characterictics

- Initially intended as safer alternative to C++
- 2010 announced by Mozilla Research
- 2015 release of first stable version
- **Compiled language**
- **Statically typed**: type known at compile time
- **Multi-paradigm**: imperative but with functional aspects
- **Low level**: offers precise control over memory and hardware resources
- **Ownership model**: enforces memory safety through strict rules



RUST **WANTS** TO BE ...

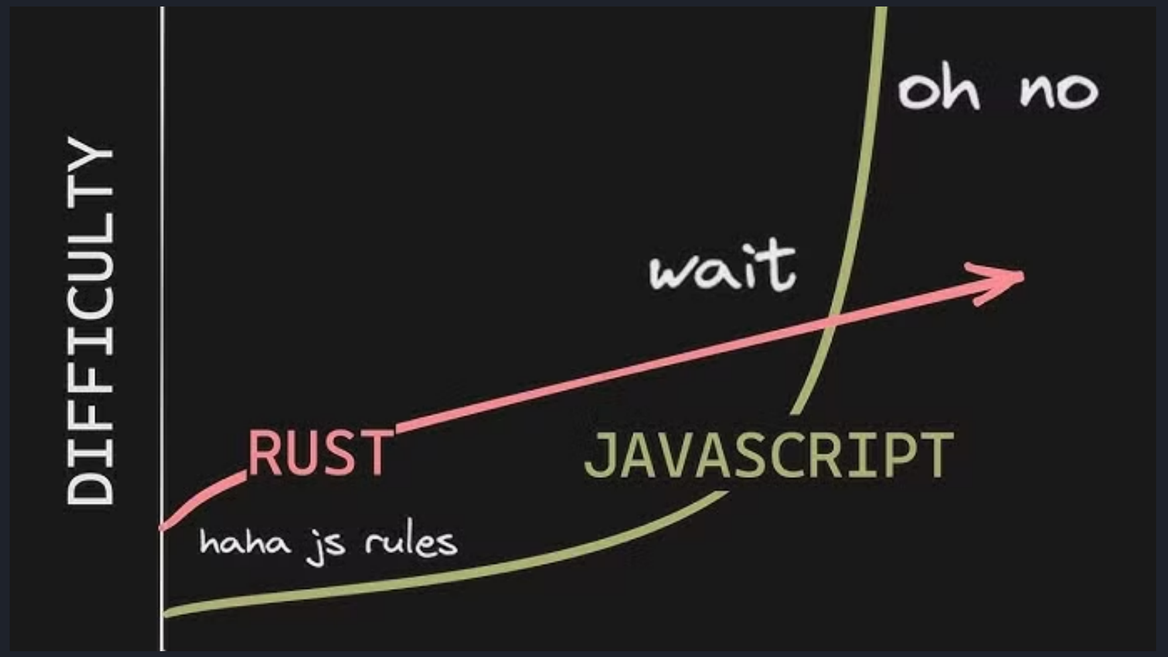
- safe & stable
- efficient
- developer- friendly

RUST **DOES NOT** TRYING TO BE ...

- scripting language
- entry programming language

Learning curve

- where are classes?
- functional principles
- what is even borrowing?



Compared to **Python** Rust:

- is a lot faster
- has lower memory use
- has static typing
- is a bit harder to pick up
- has smaller ecosystem
- throws less runtime errors

Compared to **Java** Rust:

- has lower memory use
- has no garbage collector
- provides zero-cost abstraction
- unified system for build, dependency management and testing

Compared to C++ Rust:

- has comparable memory usage and performance
- no data races
- no segmentation faults
- no null pointers
- potentially longer compile time

01 Rust Features

WHY IS RUST COOL

Memory safety

- **no dangling pointers**: references are guaranteed to be valid
- **no null pointer**: **`Option`** replaces null, ensuring handling of potentially absent values
- **no data-races**: concurrency is safe
- almost **no memory leaks**: difficult but not impossible
- much more...

OWNERSHIP

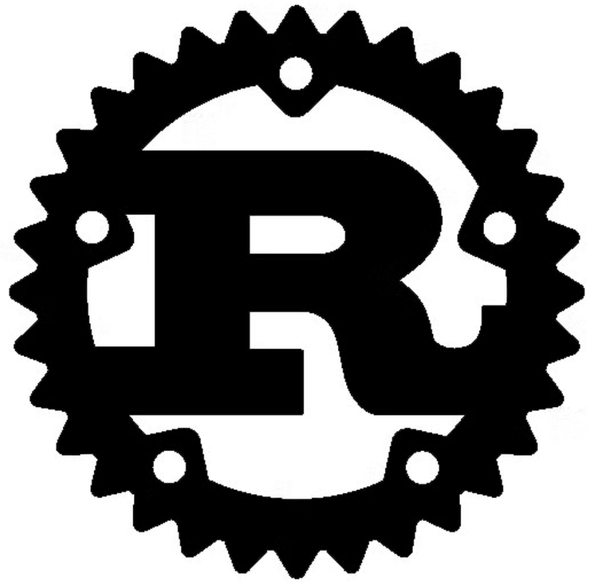
1. Each value has an owner.
2. There can only be one owner at a time.
3. When the owner goes out of scope the value drops.

BORROWING

1. At any given time, you can have *either* one mutable reference *or* any number of immutable references.
2. References must always be valid.

...much more about this next week

Who would win?



A systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety

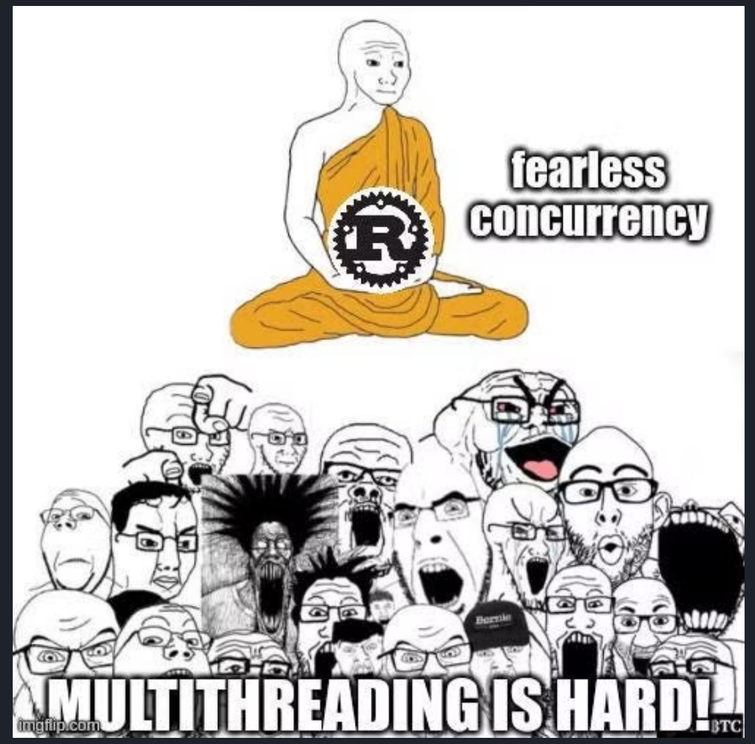
```
unsafe {
    …
}
```

Trust me.

One smol unsafe boi

Fearless concurrency

“Initially, the Rust team thought that ensuring memory safety and preventing concurrency problems were two separate challenges to be solved with different methods. Over time, the team discovered that the ownership and type systems are a powerful set of tools to help manage memory safety and concurrency problems!“



Performance

Fast ...

Performance

Fast ... comparable to C++

Tony Hoare hyperbolically apologized for inventing the null reference:

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W)

...

This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.“

Error Handling

- no null reference
- mindset: error are common and should not be exceptional
- treating errors is enforced by `Option`, `Result`
- might choose to `unwrap` → unhanded error leads to `panic!`

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

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

Error Handling

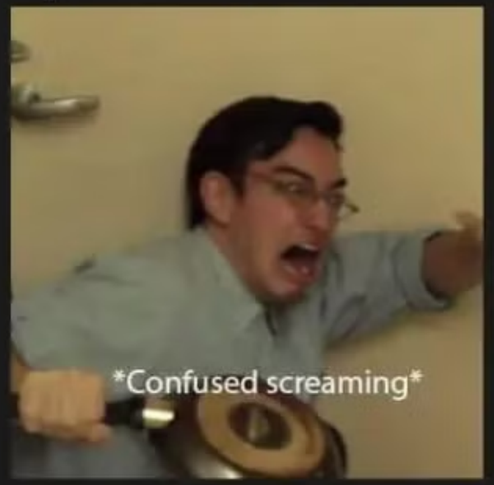
```rust
let nums = vec![3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8];
let max = numbers.iter().max(); // returns Option

match max {
    Some(&max) => println!("max element: {}", max);
    None => println!("Vector is empty.")
}
```

Compiler

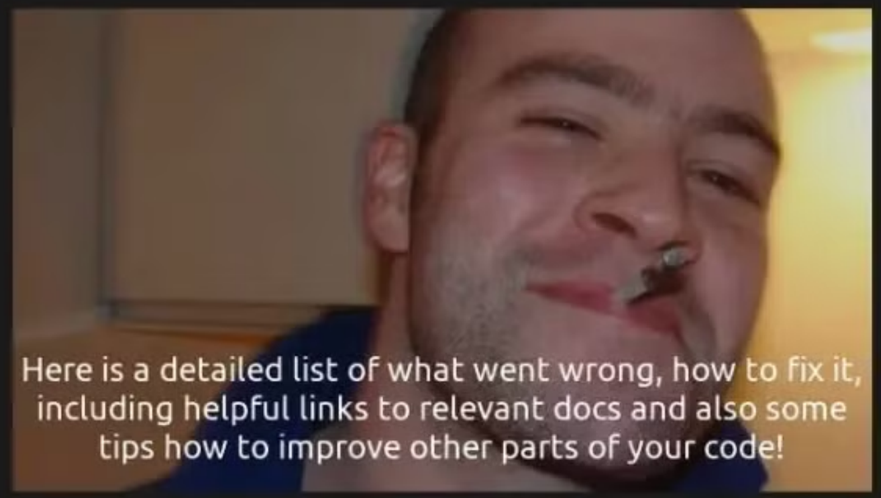
- harder to get code to compile
- if it compiles there are some guarantees and less runtime error
- nice errors with help
- compiler is you friend:)

```
error[E0384]: cannot assign twice to immutable variable `i`
 --> src/main.rs:7:3
  |
4 |     let i = 0;
  |         -
  |         |
  |         first assignment to `i`
  |         help: make this binding mutable: `mut i`
...
7 |         i += 1;
  |         ^^^^^^ cannot assign twice to immutable variable

error: aborting due to previous error; 1 warning emitted

For more information about this error, try `rustc --explain E0384`.
```

Me: Makes small mistake in C++
C++ Compiler:



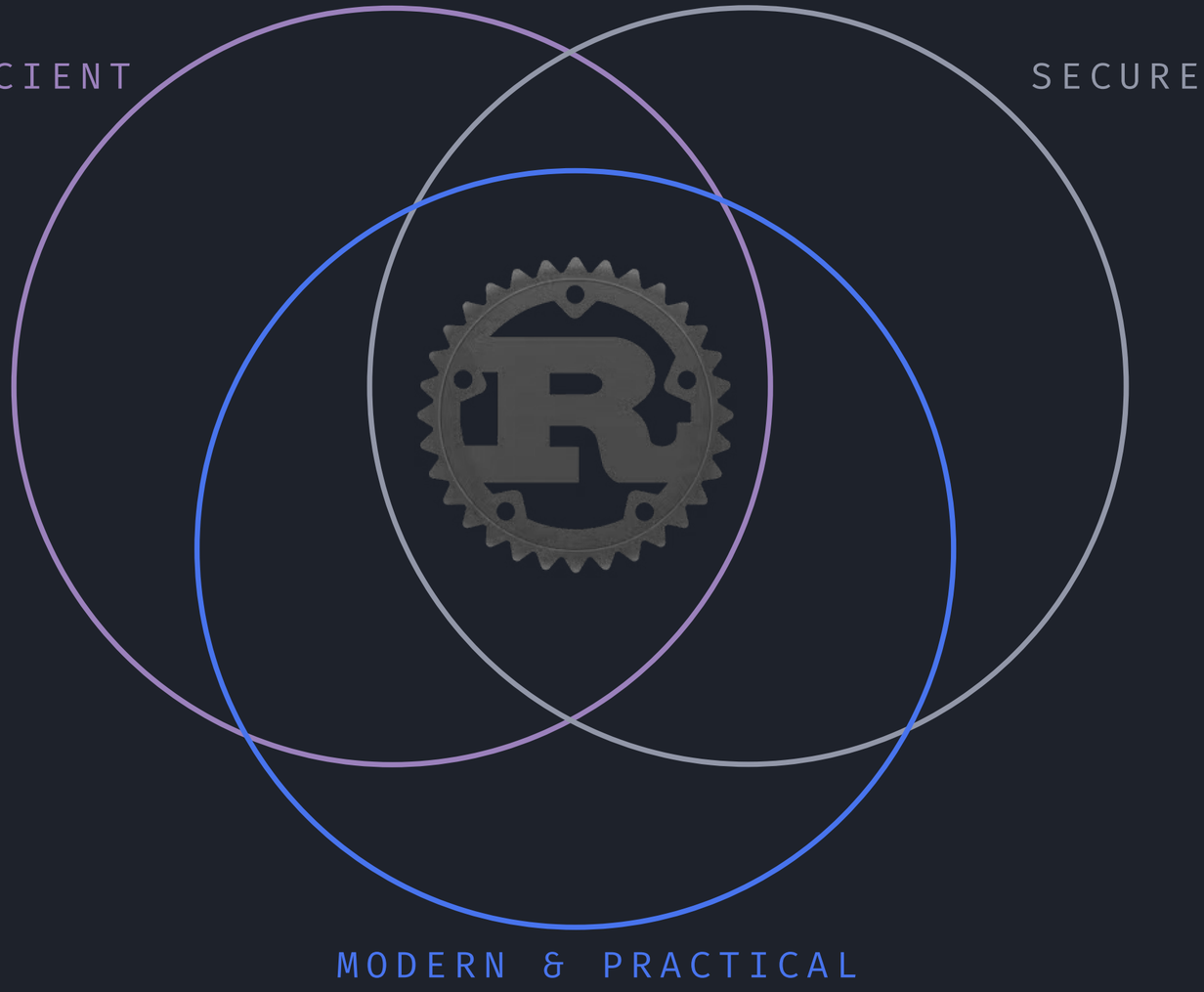
Me: Makes small mistake in Rust
Rust Compiler:



Zero-Cost Abstractions

- the abstractions you use could not have been hand-coded any better
- no hidden cost
- abstractions are resolved at compile time

Relation to Blockchain



02 Basic Syntax

HELLO WORLD!

Try Pitch

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

Is Rust Object-oriented?

- **`struct`** and **`enum`** have data, and **`impl`** blocks provide methods
- If encapsulation is a required → Rust is OO language
- If a language must have inheritance → Rust is not OO languge
  - no way to define a struct that inherits the parent struct's (without macro)
- Instead of inheritance Rust uses **trait** - similar to interfaces

Basic datatypes

- i8, i16, i32, i64, i128
- u8, u16, u32, u64, u128
- f32, f64
- bool
- char
- tuple - groups together values with a variety of types
- array - groups together values with the same type (fixed length)

String vs str

- `str`
  - stored on stack
  - immutable
- `String`
  - stored on heap
  - mutable
- no indexing into string types - UTF-8 encoding
- indexing alternatives: `chars()`, `bytes()` iterators

  → `num_string.chars().nth(i).unwrap()`

Functions

```
fn main() {
    let x = plus_one(5);

    println!("The value of x is: {x}");
}

fn plus_one(x: i32) -> i32 {
    x + 1
}

fn also_plus_one(x: i32) -> i32 {
    return x + 1;
}
```

Pattern matching

```rust
fn main() {
    let value = Some(42);

    match value {
        Some(v) => println!("Found value: {}", v),
        None => println!("No value found"),
    }

    let number = 7;
    match number {
        1..=5 => println!("Low number"),
        6..=10 => println!("High number"),
        _ => println!("Out of range"),
    }
}
```

- clear and concise syntax
- powerful → can handle literal values, variable names and wildcards...
- exhaustive: compiler confirms all possible cases are handled

Structs

```rust
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}

fn main() {
    let rect1 = Rectangle {
        width: 30,
        height: 50,
    };

    println!(
        "The area of the rectangle is {:?} square pixels.",
        rect1.area()
    );
}
```

- make up group of related objects
- unlike tuple
  - each item is named
  - struct can have methods
- cannot be partially mutable

Generics and Traits

```rust
fn largest<T: PartialOrd>(list: &[T]) -> Option<&T> {
    if list.is_empty() {
        None
    } else {
        let mut largest = &list[0];
        for item in list.iter() {
            if item > largest {
                largest = item;
            }
        }
        Some(largest)
    }
}

fn main() {
    let numbers = vec![34, 50, 25, 100, 65];
    match largest(&numbers) {
        Some(max) => println!("The largest number is: {}", max),
        None => println!("The list is empty."),
    }

    let chars = vec!['y', 'm', 'a', 'q'];
    let result = largest(&chars).unwrap();
    println!("The largest character is: {}", *result);
}
```

- **generics**: eliminate duplication by using placeholders in functions, structs, enums, and methods
- **traits**: define functionality that types can implement similarly to interfaces in OOP

03 Collections

VECTORS, HASHMAPS AND OTHER…

Vector

- vector is contiguous growable array type
- data stored on heap
- structure in memory
  - **pointer**: indicates start data in memory
  - **capacity**: amount of space allocated
    for any future elements
  - **length**: number of elements

Vector

```rust
fn main() {
    // Iterators can be collected into vectors
    let collected_iterator: Vec<i32> = (0..10).collect();

    // `vec!` macro can be used to initialize a vector
    let mut xs = vec![1, 2, 3];

    xs.push(4); // Insert new element
    collected_iterator.push(0); // !!!Immutable vectors can't grow
    let second = xs[1] // Indexing using the square brackets
    let last = xs.pop() // `pop` removes the last element

    // `Vector`s can be iterated over
    for x in xs.iter() {
        println!("> {}", x);
    }

    // Thanks to `iter_mut`, mutable `Vector`s can be iterated
    for x in xs.iter_mut() {
        *x *= 3;
    }
}
```

HashMap

```rust
fn call(number: &str) -> &str {
    match number {
        "798-1364" => "Daniel!",
        "645-7689" => "Ashley!",
        _ => "Hi! Who is this again?"
    }
}

fn main() {
    let mut contacts: HashMap<&str,&str> = HashMap::new();
    contacts.insert("Daniel", "798-1364");
    contacts.insert("Ashley", "645-7689");
    contacts.insert("Katie", "435-8291");
    contacts.insert("Robert", "956-1745");

    // Takes a reference and returns Option<&V>
    match contacts.get(&"Daniel") {
        Some(&number) => println!("Hey {} !", call(number)),
        _ => println!("Don't have Daniel's number."),
    }

    contacts.insert("Daniel", "164-6743");

    contacts.remove(&"Ashley");
    for (contact, &number) in contacts.iter() {
        println!("Calling {}: {}", contact, call(number));
    }
}
```

04 Project Structure

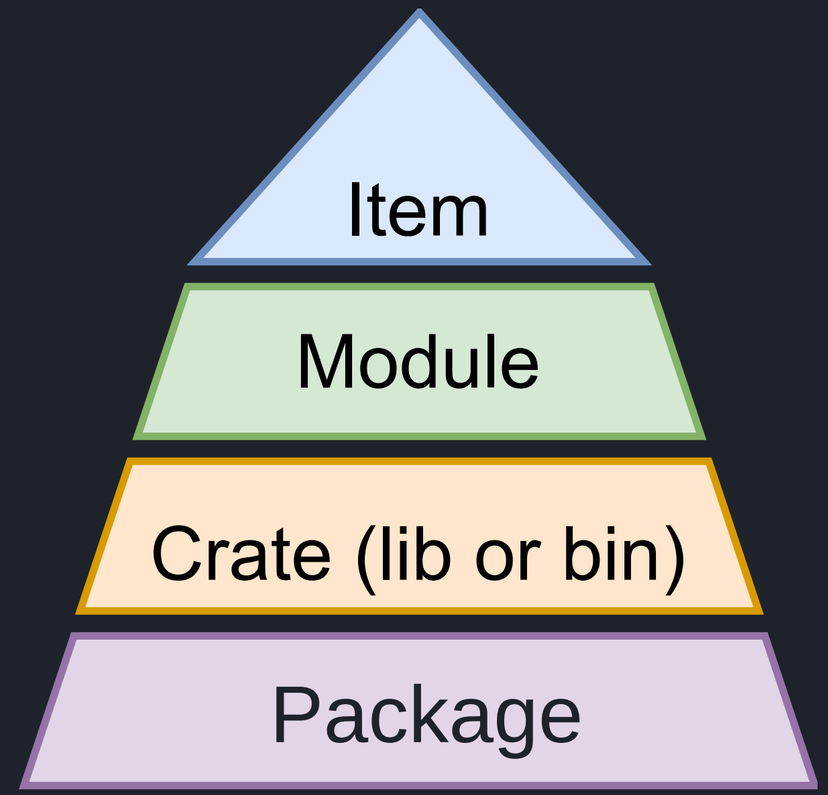
EXPLAIN WHAT THIS SECTION IS ABOUT

Cargo

- Cargo is Rust's build system and package manager (comes with the installation)
- creating a project: **`cargo new <project_name>`**
- build project: **`cargo build`**
  - compiles the code into: ./target/debug/<project_name>
  - compile with optimizations: **`cargo build —release`**
- run project (compile + run): **`cargo run`**
- cargo is also to execute unit and integration tests

Packages Crates

- **item:** piece of source code: `struct`, `fn`, etc…
- **module:** group related items into cohesive units
  - similar to namespaces
- **crate** modules that produces a library or executable
  - binary crate: programs that compile into executable
  - library crate: intended to be shared in multiple projects
- **package** is a bundle of one or more crates
  - `Cargo.toml` file describes how to build the crates



Cargo.toml

- configuration file used by Cargo specifies:
  - project metadata
  - project dependencies
  - build profiles
  - build features for conditional compilation
  - and more...

05 Hands-on

LET'S MAKE BST

Try Pitch

Getting help

THE (RUST) BOOK - Amazing resource relatively easy to read

Rust by Example - Rust explained on code snippets

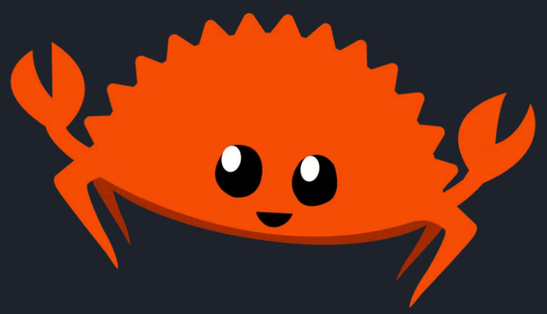
Rustilings - Small exercises to get you going

BST Task

1. Construct unbalanced BST from vector of integers
   - may assume values are unique
   - Hint: use `Option<Box<Node>>` for references
2. Add search function
3. In order traversal
4. Add function to calculate height of the tree

If you up for more

- add delete function, generics, tree iterator



https://github.com/TUM-Blockchain-Club/rust-bootcamp

Follow us

https://www.linkedin.com/company/tum-blockchain-club

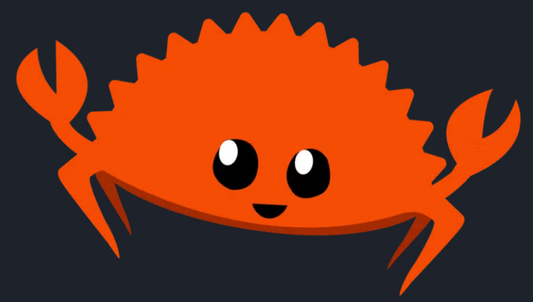
https://x.com/tbc_munich

https://warpcast.com/tumblockchain

https://www.instagram.com/tumblockchain/

https://github.com/TUM-Blockchain-Club

https://discord.gg/9Kq7Db2nwG



Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)