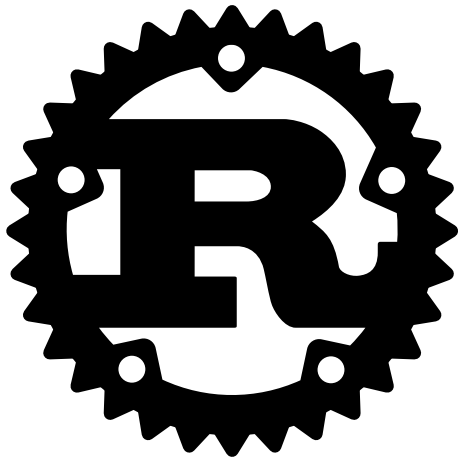


Rust Introduction

Frank Rehberger

(2025)

Agenda

- Start: 9:00
- Pause: ca 10:30 (15:min)
- Lunch: ca 12:15 (60 min)
- Pause: ca 15:00 (15 min)
- End: ca 17:00

Introduction to Rust

Day 1

Tooling, Workspace, Datatypes, Derive, Macros, File-IO

Day 2

Multithreading, Messaging, Async Programming, Web-Service, Tests

Day 3

Serialization, Libraries, Dependencies, FFI

Requirement: Attendees use their personal device with IDE and are able to install Rust from web, see: https://www.rust-lang.org/tools/install

Demo Project

Github

https://github.com/frehberg/rust-demo-project.git

Terminology

Terminology

Framework

Rust

Language Syntax and Compiler rustc.

rustup-tool

Rust update tool installing rustc, cargo, clippy, etc.

Package

Versioned deliverable of crates at https://crates.io/ or github.

Cargo

Build tool reading Cargo.toml, downloading crates and compiling source.

Terminology

Cargo Features

Build-Script

Project file ./build.rs performing pre-processing.

Cargo Plugins

Extensions of Cargo, eg. Clippy or Workspaces.

Overview

- 2015 - Release of version 1.0, current ver 1.84
- System programming.
- Memory-safety.
- No runtime, just rust on top of bare-metal.
- Feature Editions 2018, 2024, etc.
- Bundled with Cargo Build Tool + version + X-comp.
- Releasing stable and bi-weekly unstable version.

```
$ rustup  toolchain list; rustup update
```

- Tool managing installation of target X-compilers

```
$ rustup target list
```

Basic Syntax

Syntax is similar to other curly-brace languages, like C++ or Scala

Data is organized into structs, enums (algebraic data types), tuples, and primitives

Safety

In safe* Rust, it is impossible to get segfaults, memory corruption, dangling pointers, or data races.

This is achieved with Rust's ownership system, which requires that all data have a single owner on the stack

Owners can lend out their data via "borrowing", which is checked at compile time by Rust's "Borrow Checker"

But I only ever write single-threaded programs, I'm never going to run into any of those bugs!

```
for(auto it=v.begin(); it!=v.end(); ++it) {
  if (*it < 5) {
    v.push_back(5 - *it);
  }
}
```

Ownership & Borrowing

Rust has move semantics by default for most complex data types, which prevents having multiple variables having access to the same variable

When you assign a variable to another or pass it to function as an argument, it is no longer accessible

Traits

Rust implements OO via a system called Traits

Similar to Type Classes in Haskell (and different from most OO), Traits do not have any associated data

Independend Projects at Github

Rust Webpage https://www.rust-lang.org/

Rust

https://github.com/rust-lang/rust

Crates.io

https://github.com/rust-lang/crates.io

Cargo

https://github.com/rust-lang/cargo

Rust-Clippy

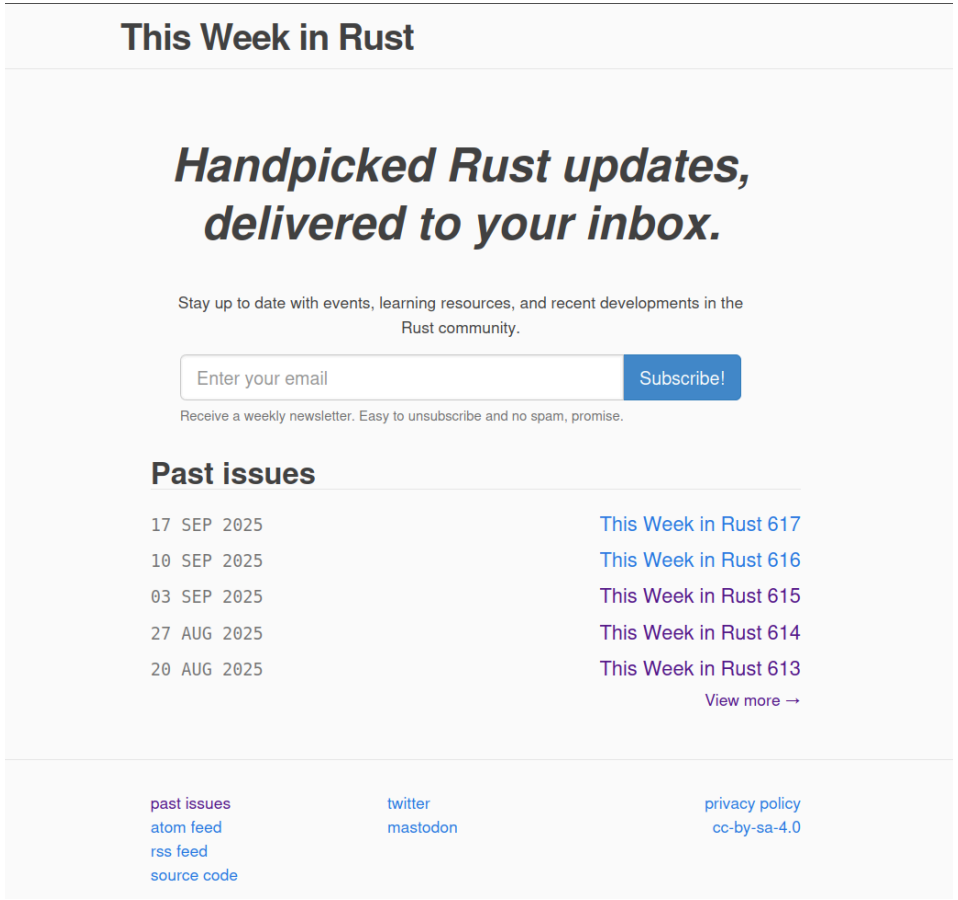
https://github.com/rust-lang/rust-clippy

RFCs

https://github.com/rust-lang/rfcs

Weekly Rust News

Community new https://this-week-in-rust.org/.



Weekly Rust News

Updates from Rust community

Updates from Rust Community

Official

- Crossing the streams: Project + Foundation
- Rust compiler performance survey 2025 results

Newsletters

- This Month in Rust OSDev: August 2025

Project/Tooling Updates

- Rust Automod VSCode Extension - Automates creation and management of `mod.rs` files
- Now available: Rust SDK for Google Cloud

Observations/Thoughts

- Protecting Rust against supply chain attacks
- The unreasonable effectiveness of modern sort algorithms
- Improving state machine code generation
- [video] How Rust won: the quest for performant, reliable software
- [video] Rust for Everyone!

Rust Walkthroughs

- Axum Backend Series - Introduction

Miscellaneous

Weekly Rust News

Updates from Rust project, RFC updates.

Updates from the Rust Project

390 pull requests were merged in the last week

Compiler

- fix drop scope for `super let` bindings within `if let`
- stabilize c-style varargs for sysv64, win64, efiapi, aapcs

Library

- add exact bitshifts
- constify impl Try for ControlFlow
- fix path str eq
- single buffer for exponent fmt of integers
- stabilize `path_add_extension`
- implement WASIp2-specific stdio routines
- start supporting WASIp2 natively

Cargo

- optimize Cargo with LTO
- `fix(manifest)`: Report script manifest errors for the right line number
- fix: switch from --nocapture to --no-capture
- render individual compilation sections in `--timings` pipeline graph

Rustdoc

Weekly Rust News

Virtual and local events in Americas, Europe, and Asia

Upcoming Events

Rusty Events between 2025-09-10 - 2025-10-08 🦀

Virtual

- 2025-09-11 | Virtual (Berlin, DE) | Rust Berlin
  - **Rust Hack and Learn**
- 2025-09-11 | Virtual (San Diego, CA, US) | San Diego Rust
  - **San Diego Rust September 2025 Online Meetup**
- 2025-09-14 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup
  - **Rust Readers Discord Discussion: Rust Atomics and Locks**
- 2025-09-15 | Virtual (Charlottesville, VA, US) | Charlottesville Rust Meetup
  - **Setup Tock OS in a virtual environment (online) - prep for Sep 17**
- 2025-09-16 | Virtual (Washington, DC, US) | Rust DC
  - **Mid-month Rustful**
- 2025-09-17 | Virtual (Vancouver, BC, CA) | Vancouver Rust
  - **Rust Study/Hack/Hang-out**
- 2025-09-18 | Virtual (Nürnberg, DE) | Rust Nuremberg
  - **Rust Nürnberg online**
- 2025-09-23 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup
  - **Fourth Tuesday**
- 2025-09-24 | Virtual (Lima, PE) | Rust Perú
  - **Rust Perú Online Meetup: Macros de Assembly y Backends con SurrealDB + Axum**

Rust Book

https://doc.rust-lang.org/book

The Rust Programming Language

by Steve Klabnik, Carol Nichols, and Chris Krycho, with contributions from the Rust Community

This version of the text assumes you're using Rust 1.85.0 (released 2025-02-17) or later with `edition = "2024"` in the Cargo.toml file of all projects to configure them to use Rust 2024 edition idioms. See the "Installation" section of Chapter 1 to install or update Rust.

The HTML format is available online at https://doc.rust-lang.org/stable/book/ and offline with installations of Rust made with `rustup`; run `rustup doc --book` to open.

Several community translations are also available.

This text is available in paperback and ebook format from No Starch Press.

🚨 **Want a more interactive learning experience? Try out a different version of the Rust Book, featuring: quizzes, highlighting, visualizations, and more:** https://rust-book.cs.brown.edu

Cargo Book

https://doc.rust-lang.org/cargo/

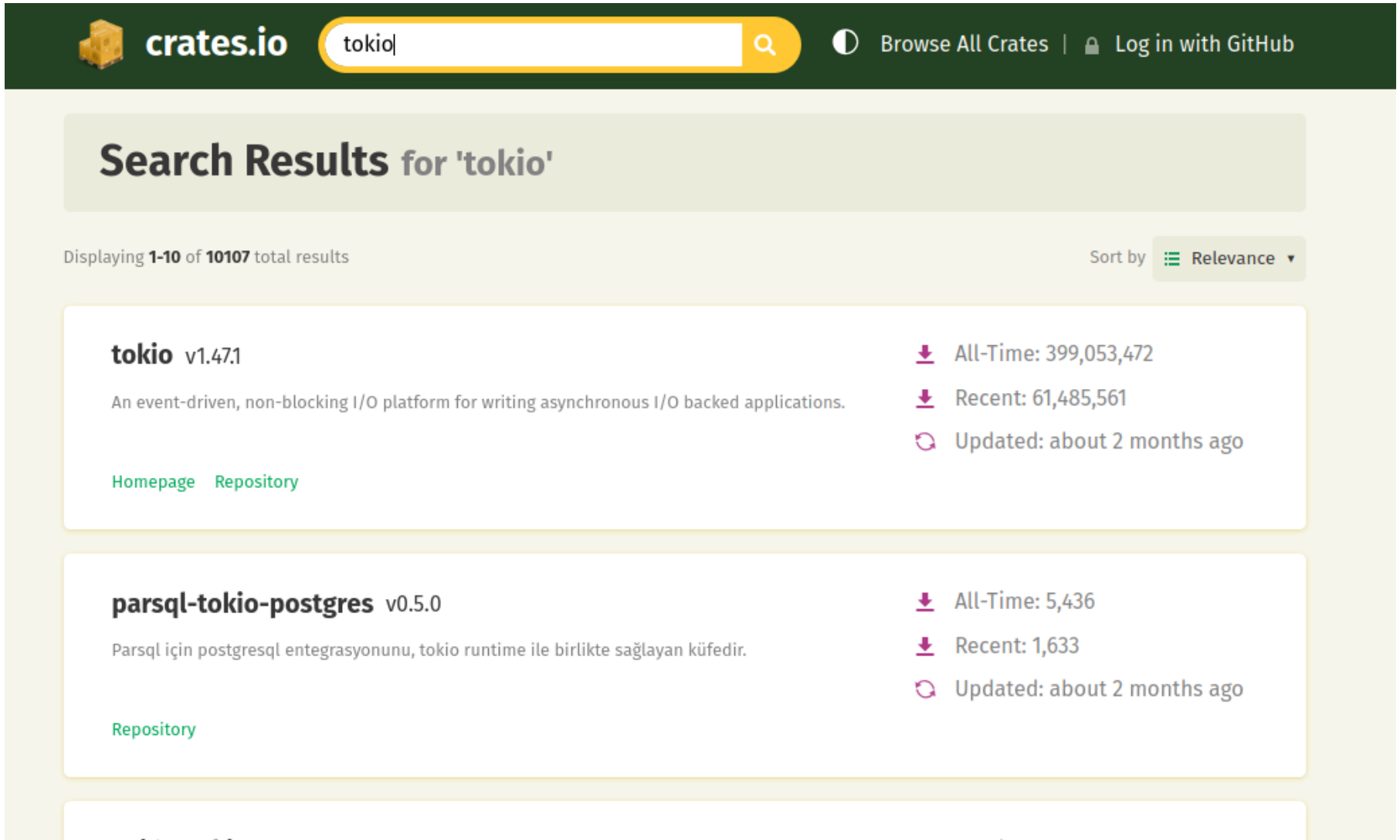
The Cargo Book

Cargo is the Rust *package manager*. Cargo downloads your Rust package's dependencies, compiles your packages, makes distributable packages, and uploads them to crates.io, the Rust community's *package registry*. You can contribute to this book on GitHub.

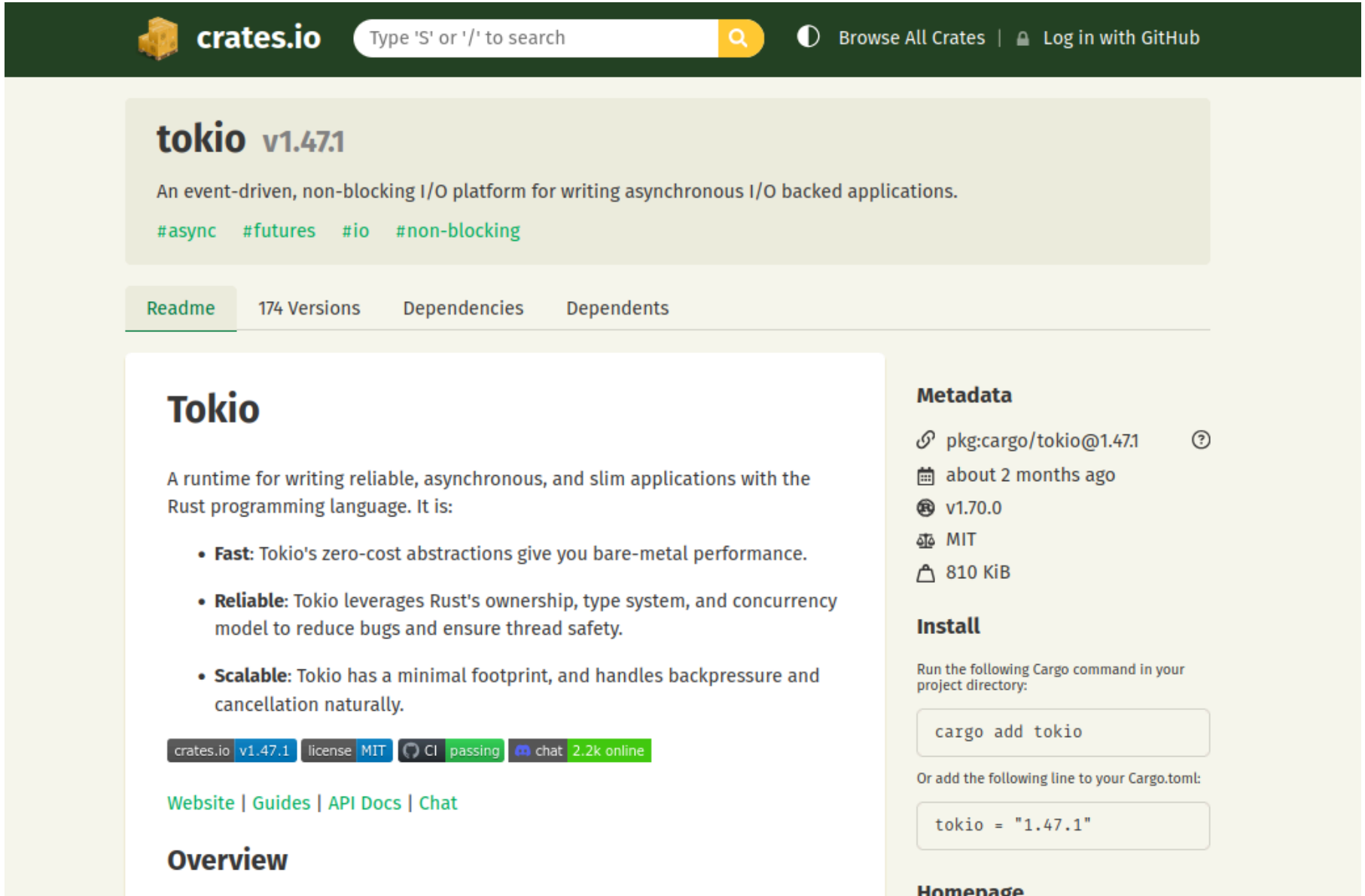
Packages/Crates

Package Filter

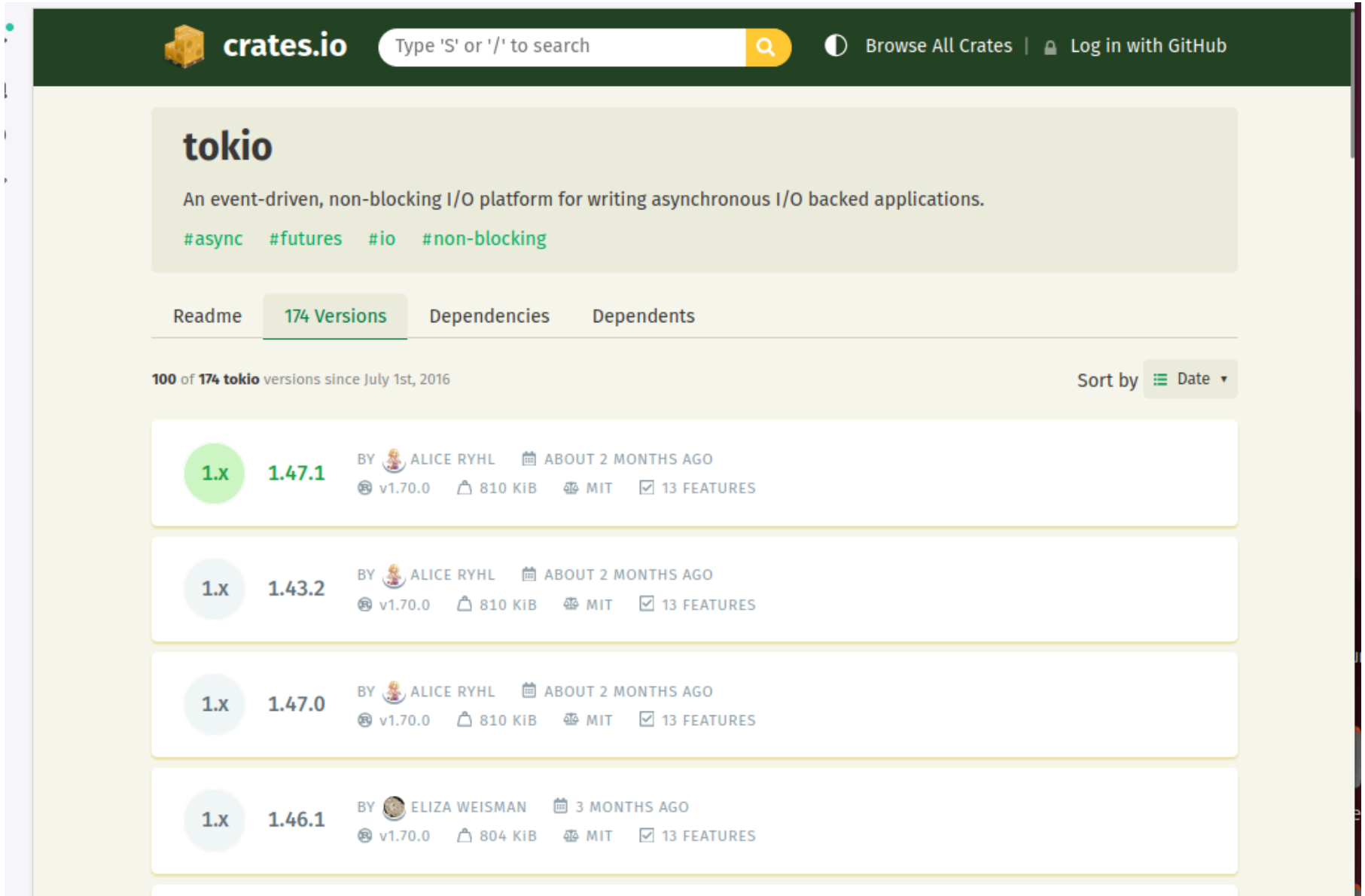
Located at https://crates.io/, formerly https://crates.rs



Package View



Package Versions View



Cargo Tool

Cargo Build Functionality

See options using `cargo COMMAND -h`

build

Compile a package.

check

Check a local package and all of its dependencies for errors.

doc

Build a package's documentation.

run

Build and run an executable.

install

Install Cargo plugin/extension.

Cargo Extensions

cargo-workspaces: manage multi-member workspaces

```
$ cargo install cargo-workspaces
```

cargo-expand: expand macros

```
$ cargo install cargo-expand
```

Using cargo default functionality

```
$ cargo new my-project --bin
    Creating binary (application) `my-project` package
note: see more `Cargo.toml` keys and their definitions at https:/
```

Creating package structure incl Cargo.toml

```
my-project/
├── Cargo.toml
└── src
└── main.rs
```

Cargo.toml

```
[package]
name = "my-project"
version = "0.1.0"
edition = "2024"

[dependencies]
```

Programming Concepts

Variables

Variables and let-expression

File src/main.rs

```
fn main() {
    let mut x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```

Variable Shadowing

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

    let x = x + 1;

    {
        let x = x * 2;
        println!("The value of x in the inner scope is: {x}");
    }

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

Variable Redefinition

```
let spaces = "    ";
let spaces = spaces.len();
```

Programming Concepts

Data Types

Integer Types

Length	Signed	Unsigned
8-bit	i8	u8
16-bit	i16	u16
32-bit	i32	u32
64-bit	i64	64
128-bit	i128	u128

Note: Overflow is checked for `+` `-` `*` `/` in release builds, instead use `unchecked_OP` or `wrapping_OP`

```
10u8.wrapping_mul(12)
```

Number Literals

Number Literals	Example
Decimal	98_222
Hex	0xff
Octal	0o77
Binary	0b1111_0000
Bytes	b'A'

Floating-Point Types

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

    let y: f32 = 3.0; // f32
}
```

Numerical Operations

```
// addition
let sum = 5 + 10;

// subtraction
let difference = 95.5 - 4.3;

// multiplication
let product = 4 * 30;

// division
let quotient = 56.7 / 32.2;
let truncated = -5 / 3; // Results in -1

// remainder
let remainder = 43 % 5;
```

Boolean Type

```
  fn main() {
    let t = true;

    let f: bool = false; // with explicit type annotation
}
```

Character Type (Utf8)

```
fn main() {
    let c = 'z';
    let z: char = 'ℤ'; // with explicit type annotation
    let heart_eyed_cat = '😻';
}
```

Tuple Types

```
let tup: (i32, f64, u8) = (500, 6.4, 1);

// same using inference
let tup = (500, 6.4, 1);

let (x, y, z) = tup;

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

Tuple Types

Special Nothing Type

```
let none_type = ();
// cannot be printed
```

Array Types

```
let a: [i32; 5] = [1, 2, 3, 4, 5];

let months = ["January", "February", "March", "April", "May",
              "August", "September", "October", "November", "

let a = [3; 5];

// accessing
let first = a[0];
let second = a[1];
```

String Types

Basic str& Reference vs String Object.

```
fn main() {
        let basic_string : &'static str =
            "Static String, length encoded";
        let dynamic_string_object: String =
            format!("{} {}", "Dynamic", "String");
        let basic_string_with_lifetime : &str =
            &dynamic_string_object;

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

Basic String are simple Char arrays, UTF8 encoded

Raw Strings Types

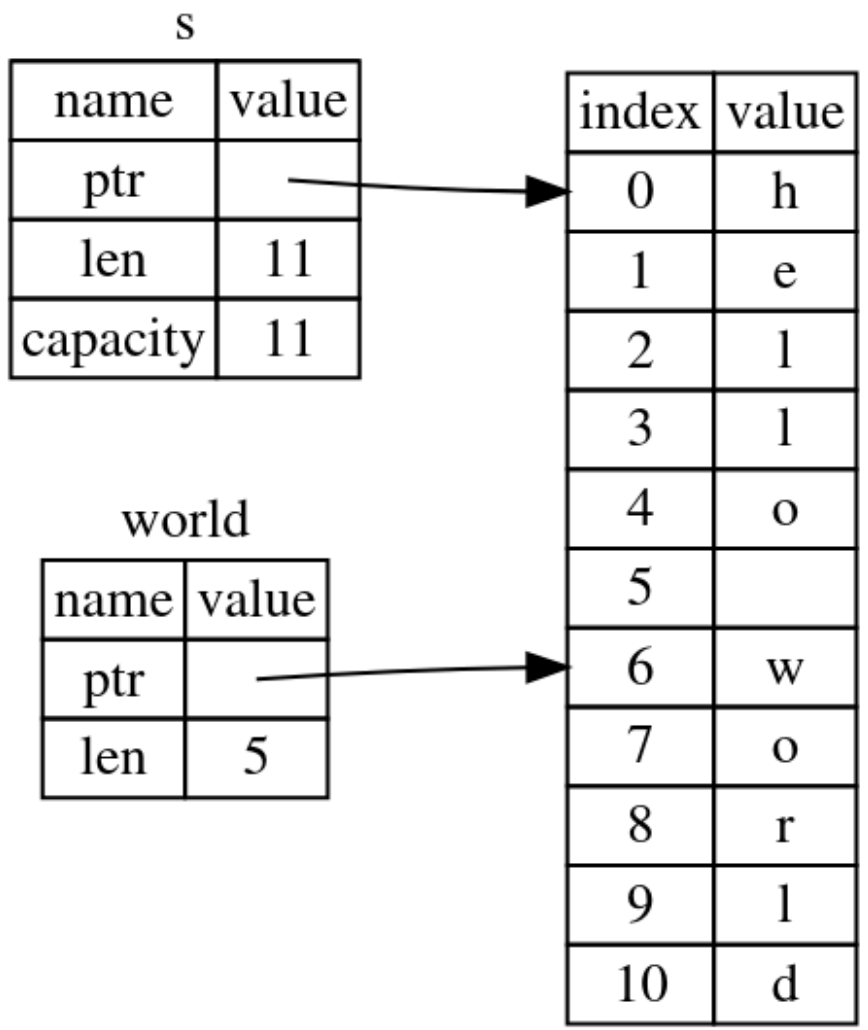
Raw String are u8 bytes arrays.

```
fn main() {
        let raw_byte_string: &[u8; 48] =
           b"HTTP/2 200\ncontent-type: text/css; charset=utf-8";
        let raw_byte_string_slice = &raw_byte_string[0..5];

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

String Slices

Slices are length encoded.

```
fn main() {
      let s = String::from("hello world"

      let hello: &str = &s[0..5];
      let world: &str = &s[6..11];
}
```



Types Example

```
use std::io;
fn main() {
    println!("Please enter an array index.");
    let mut index = String::new();
    io::stdin()
        .read_line(&mut index)
        .expect("Failed to read line");
    let index: usize = index
        .trim()
        .parse()
        .expect("Index entered was not a number");
    println!("The value is {index}");
}
```

Programming Concepts

Functions

Functions & Parameters

```
fn main() {
    another_function(5);
}

fn another_function(x: i32) {
    println!("The value of x is: {x}");
}
```

Functions & Parameters

```
fn main() {
    another_function(5);
}

fn another_function(x: i32) {
    println!("The value of x is: {x}");
}
```

Expresssions

```
fn main() {
    let y = {
        let x = 3;
        x + 1
    };

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

Return Values

Expressions vs return statement

```
// expression
fn five() -> i32 {
    5
}
// using return statement
fn six() -> i32 {
    return 5;
}

fn main() {
    let x = five();

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

Lambda expressions

```
// global constant, static-lifetime
const BASE: i32 = 100;

fn main() {
    let offset = | d: i32 | -> i32 { BASE + d };

    println!("The offset value is: {}", offset(5));
}
```

Programming Concepts

Macros

Meta-Programming, transforming parser's token stream.

- Custom derive-Macros.
- Attribute-like macros.
- Function-like macros (eg vec!).

Function like macros

Create vector with builtin macro vec!

```
let v: Vec<u32> = vec![1, 2, 3];
```

```
let v: Vec<u32> = <[_]>::into_vec(
   ::alloc::boxed::box_new([1, 2, 3]));
```

```
#[macro_export]
macro_rules! vec {
    () => { ... };
    ($elem:expr; $n:expr) => { ... };
    ($($x:expr),+ $(,)?) => { ... };
}
```

Custom derive-Macros

Transformation by macro expansion

```
#[derive(Debug)]
struct DataT {
    flag: bool,
    val: isize,
}
```

```
// These macros
// are a bit
// more complex,
// as compiler plugin.
```

```
#[automatically_derived]
impl ::core::fmt::Debug for DataT {
    #[inline]
    fn fmt(&self, f: &mut ::core::fmt::Formatter)
        -> ::core::fmt::Result {
         ::core::fmt::Formatter::debug_struct_field2_fi
              f,
              "DataT",
              "flag",
              &self.flag,
              "val",
              &&self.val,
         )
    }
}
```

Programming Concepts

Comments and API Docs

Comments & API Docs

```
//! First line as title of front page.
//!
//! Futher lines describing
//! the application.

/// This module makes it easy.
pub mod easy {

    /// Use the abstraction function to do this specific thing.
    pub fn abstraction() {
         // simple comment
         let x = 7;
         ...
    }
}
```

API Doc Generation

```
cargo doc
Documenting libc v0.2.175
// snip
Documenting mylib1 v0.0.0 (./project/mylib1)
   Finished `dev` profile [unoptimized + debuginfo] target(s)
  Generated ./project/target/doc/mylib1/index.html
```

Programming Concepts

Control Flow

If Expression

```
fn main() {
    let number = 6;

    if number % 4 == 0 {
        println!("number is divisible by 4");
    } else if number % 3 == 0 {
        println!("number is divisible by 3");
    } else if number % 2 == 0 {
        println!("number is divisible by 2");
    } else {
        println!("number is not divisible by 4, 3, or 2");
    }
}
```

if in let Expressions

```
fn main() {
    let condition = true;
    let number = if condition { 5 } else { 6 };

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

let in if Expressions

type enum Data { Foo(i32), Bar(u32) };

```
fn do_something(data: &Data) {
    if let Data::Foo(val) = data {
        do_foo(val);
    }
}
```

loop Expressions

```
fn main() {
    let mut counter = 0;

    let result = loop {
        counter += 1;

        if counter == 10 {
            break counter * 2;
        }
    };

    println!("The result is {result}");
}
```

Conditional Loops with while

```
fn main() {
    let mut number = 3;

    while number != 0 {
        println!("{number}!");

        number -= 1;
    }

    println!("LIFTOFF!!!");
}
```

Looping Through a Collection with for

```
fn main() {
    for number in (1..4).rev() {
        println!("{number}!");
    }
    println!("LIFTOFF!!!");
}
```

Ownership

How a Rust program manages memory

- Each value in Rust has an owner.
- There can only be one owner at a time.
- When the owner goes out of scope, the value will be dropped.
- By default values are immutable.

Variable Scope

Base string in static memory.

```
{                       // s is not valid here, not yet declared
    let s = "hello"; // s is valid from this point forward

    // do stuff with s
}                      // this scope is now over, no longer valid
```

Note: s is static str-ref

```
{
    let s: &'static str = "...";
}
```

Complex Object (String)

Immutable String

```
let s = String::from("hello");
```

Complex Object (String)

Mutable String, allocated on heap.

```
let mut s = String::from("hello");

s.push_str(", world!"); // push_str() appends a literal to a

println!("{s}"); // this will print `hello, world!`
```

Variables and Data - Copy

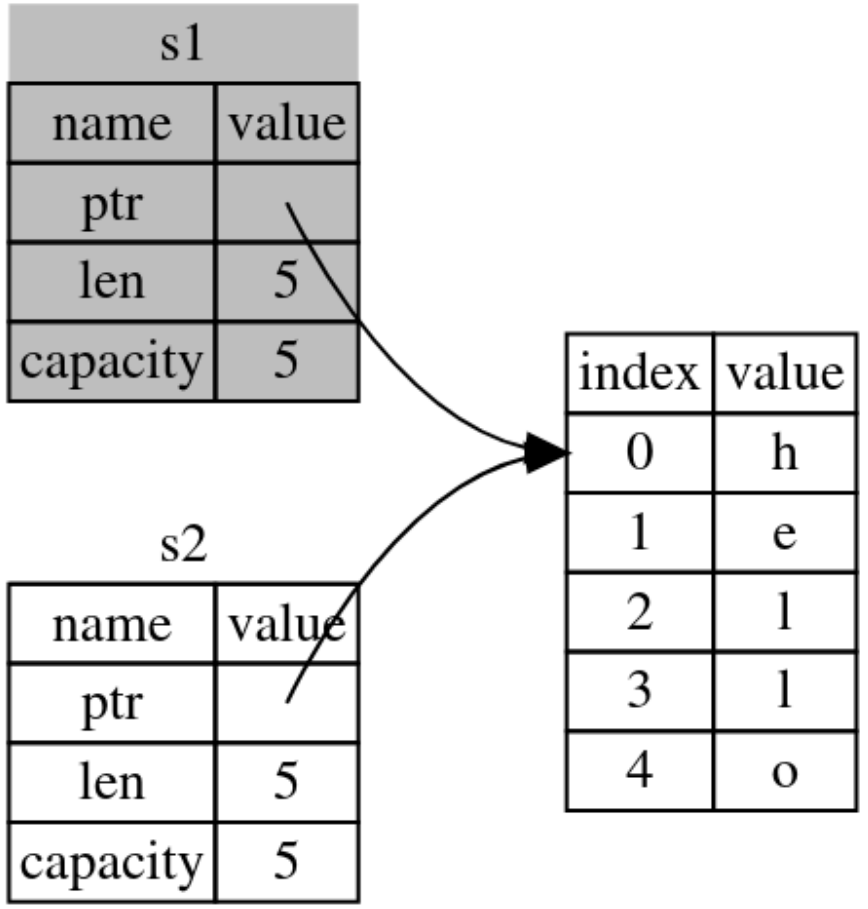
Integers have known size and are copied.

```
let x = 5; // integer has known size
let y = x;
// x and y accessible
```

Variables and Data - Move

Size unknown.
No Copy implemented.
Ownership moved.

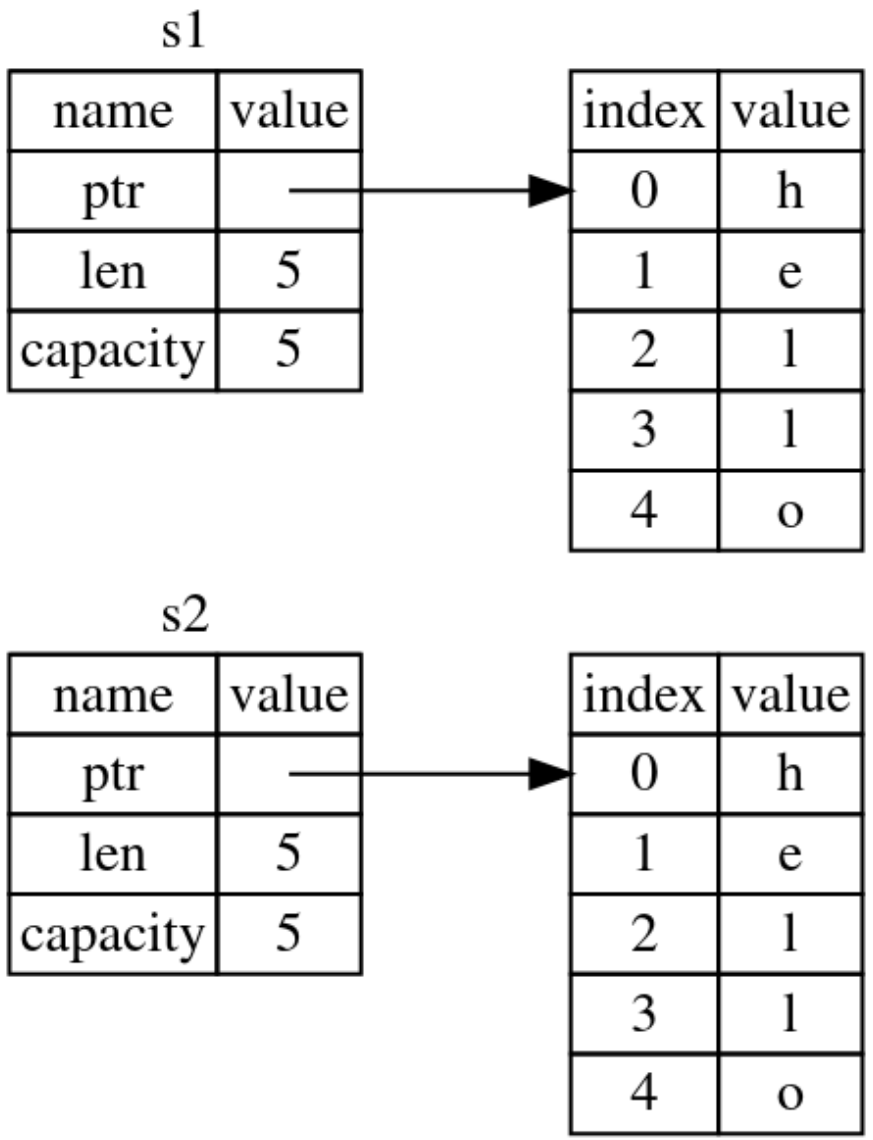
```
let s1: String = String::from("hello");
let s2 = s1;
// s1 no longer accessible,
// ownership moved to s2
```



Variables and Data - Clone

Perform deep copy.

```
let s1 = String::from("hello");
let s2 = s1.clone();

println!("s1 = {s1}, s2 = {s2}");
```



Return Values and Scope

```
fn create() -> String { String::from("hello") }

fn take_and_return(s: String) -> String { s }

fn main() {
    let s1 =
      create();    // return value moved,
                   // assigned to s1
    let s2 =
       takes_and_returns(s1); // s1 is moved into
                              // fn returns some String
                              // assigned to s2
}
```

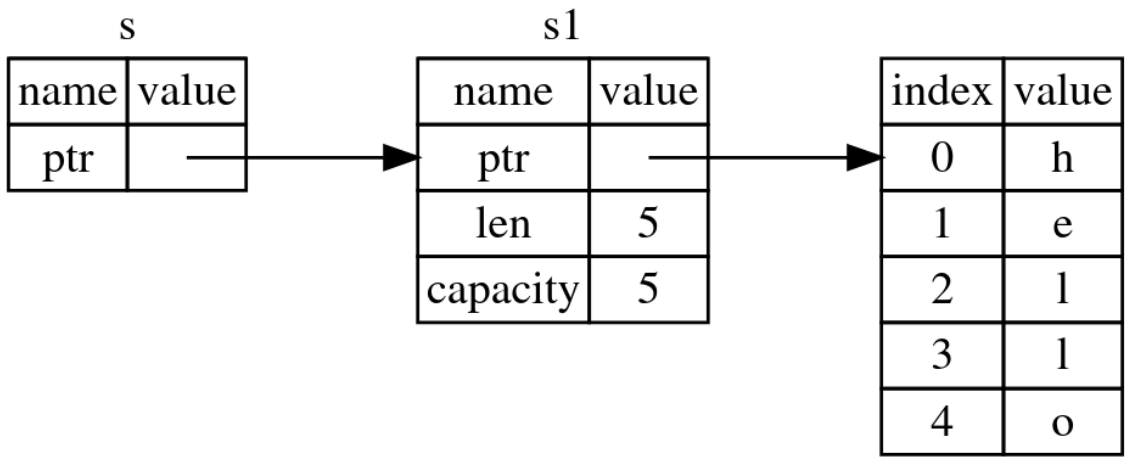
References & Borrowing

How a Rust program manages memory

- Reference is guaranteed to point to a valid value.
- Mutable values can be referenced once only.
- References are bound to lifetime of value.
- Global values must be const or protected (lock).

References & Borrowing - Immutable

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

    let len = calculate_length(&s1);

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

fn calculate_length(s: &String) -> usize {
    s.len()
}
```



References & Borrowing - Immutable

Immutable Static

```
1   const UUID: &'static str = "919950d0-9..";
2 fn main() {
3       let uuids: Vec<&'static str> = vec![UUID];
4 ...
5 }
```

References & Borrowing - Mutable

```
1 fn main() {
2     let s = String::from("hello");
3
4     change(&mut s);
5 }
6
7 fn change(some_string: &mut String) {
8     some_string.push_str(", world");
9 }
```

Global Values

static

A possibly mutable variable with 'static lifetime

const

An unchangeable value (the common case).

Lazy Static Variable

```
use std::sync::LazyLock;

static LAZY: LazyLock<i32>
                = LazyLock::new(|| {
    println!("initializing");
    92
});
fn main() {
    println!("ready");
    println!("{}", &*lazy);
}
```

Const Variable

```
const CONST_VAL: i32 = 92;
fn main() {
  println!("{}", CONST_VAL);
}
```

Structures

Grouping Data

- Visibility of fields defaults to private.
- Outside of module, only 'pub' fields may be accessed.
- Function implementations may read/modify fields.

Defining and Instantiating Structs

```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}

fn main() {
    let user1 = User {
        active: true,
        username: String::from("someusername123"),
        email: String::from("someone@example.com"),
        sign_in_count: 1,
    };
}
```

Function Returning Struct

```
1  struct User {
2      active: bool,
3      username: String,
4      email: String,
5      sign_in_count: u64,
6  }
7
8  fn build_user(email: String, username: String) -> User {
9      User {
10         active: true,
11         username: username,
12         email: email,
13         sign_in_count: 1,
14     }
15 }
```

Using the Field Init Shorthand

```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}

fn build_user(email: String, username: String) -> User {
    User {
        active: true,
        username,
        email,
        sign_in_count: 1,
    }
}
```

Using the Update Synax

```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}
fn main() {
    let user1 = User {..};
    // --snip--

    let user2 = User {
        email: String::from("another@example.com"),
        ..user1
    };
}
```

Using Tuple Structs

Without Named Fields

```
1 struct Color(i32, i32, i32);
2 struct Point(i32, i32, i32);
3
4 fn main() {
5     let black = Color(0, 0, 0);
6     let origin = Point(0, 0, 0);
7 }
```

Unit-Like Structs

Without Named Fields, used for stateless objects.

```
1 struct AlwaysEqual;
2
3 fn main() {
4     let subject = AlwaysEqual;
5 }
```

Tuple Struct - Example

```
1  fn main() {
2      let rect1 = (30, 50);
3
4      println!(
5          "The area of the rectangle is {} square pixels.",
6          area(rect1)
7      );
8  }
9
10 fn area(dimensions: (u32, u32)) -> u32 {
11     dimensions.0 * dimensions.1
12 }
```

Defining Methods with impl

The &self is a short for 'self: &Self'.

```
#[derive(Debug)]
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()
    );
}
```

Defining Methods with impl

More parameters.

```
#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}
impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
    fn can_hold(&self, other: &Rectangle) -> bool {
        self.width > other.width && self.height > other.height
    }
}
```

Defining Methods with impl

Associated Functions - May be used as Constructors

```
#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}
impl Rectangle {
    fn square(size: u32) -> Self {
        Self {
            width: size,
            height: size,
        }
    }
}
```

Defining Methods with impl

Multiple impl blocks allowed.

```
1  impl Rectangle {
2      fn area(&self) -> u32 {
3          self.width * self.height
4      }
5  }
6
7  impl Rectangle {
8      fn can_hold(&self, other: &Rectangle) -> bool {
9          self.width > other.width && self.height > other.height
10     }
11 }
12
```

Enums and Pattern Matching

Defining an Enum

Enumerate all possible variants.

```
1 enum IpAddr {
2     V4,
3     V6
4 }
5
```

Defining an Enum - Tuple-Type

Enumerate all possible variants.

```
1 enum IpAddr {
2     V4(u8, u8, u8, u8),
3     V6(String),
4 }
5
6 let home = IpAddr::V4(127, 0, 0, 1);
7
8 let loopback = IpAddr::V6(String::from("::1"));
```

Defining an Enum - Struct-Type

Enumerate all possible variants.

```
1 enum IpAddr {
2     V4{ addr: [u8; 4],},
3     V6{ addr: String, },
4 }
5
6 let home = IpAddr::V4{addr: [127, 0, 0, 1]};
7
8 let loopback = IpAddr::V6{addr: String::from("::1"), };
```

Option Type

Avoid "Null References: The Billion Dollar Mistake"

```
// declared in std::option
enum Option<T> {
    None,
    Some(T),
}
```

```
use std::option::Option;
fn main()  {
    let some_number = Some(5);
    let some_char = Some('e');

    let absent_number: Option<i32> = None;
}
```

Note:The <T> is a generic type parameter.

Result Type

```
// declared in std::result
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```
use std::fs::File;
use std::io::Error;
use std::result::Result;
fn main() {
    let result: Result<std::fs::File, std::io::Error> =
            File::open("hello.txt");
    match result {
        Ok(fout) => { /* write to open file */ },
        Err(error) => {
            println!("Failed to open file {error}");
        }
    }
// snip
}
```

Note:The <T> is a generic type parameter.

The match Control Flow Construct

```
enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter,
}
fn value_in_cents(coin: Coin) -> u8 {
    match coin {
        Coin::Penny => { println!("Lucky penny!");  1  },
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter => 25,
    }
}
```

Pattern Bindings

```
1
2  enum Coin { Penny, Nickel, Dime, Quarter(String), }
3
4  fn value_in_cents(coin: Coin) -> u8 {
5      match coin {
6          Coin::Penny => 1,
7          Coin::Nickel => 5,
8          Coin::Dime => 10,
9          Coin::Quarter(state) => {
10             println!("State quarter from {state:?}!");
11             25
12         }
13     }
14 }
```

Implement Functions for Enum

```
1  enum Coin { Penny, Nickel, Dime, Quarter(String), }
2
3  impl Coin {
4      fn value_in_cents(&self) -> u8 {
5          match self {
6              Coin::Penny => 1,
7              Coin::Nickel => 5,
8              Coin::Dime => 10,
9              Coin::Quarter(_state) => 25,
10         }
11     }
12 }
```

Note: variable `_state` marked as unused

Matching with Option

```
 1 fn plus_one(x: Option<i32>) -> Option<i32> {
 2     match x {
 3         _ => None,
 4         Some(i) => Some(i + 1),
 5     }
 6 }
 7 // snip ..
 8 let five = Some(5);
 9 let six = plus_one(five);
10 let none = plus_one(None);
```

Concise Control Flow with if let and let else

```
let config_max = Some(3u8);
match config_max {
    Some(max) => println!("The maximum is configured to be {max
    _ => (),
}
```

shorter, but loosing exhaustive check of all variants

```
let config_max = Some(3u8);
if let Some(max) = config_max {
    println!("The maximum is configured to be {max}");
}
```

Type Aliases

Keyword `type` defines a type alias.

```
1 type CoinBag = Vec<coin>;
2  </coin>
```

Hands On 1

- Create workspace with Cargo.
- Implement this function returning a None in case x==7 otherwise ident.

```
1  fn do_call(x: i32) -> Option<i32> ;
```

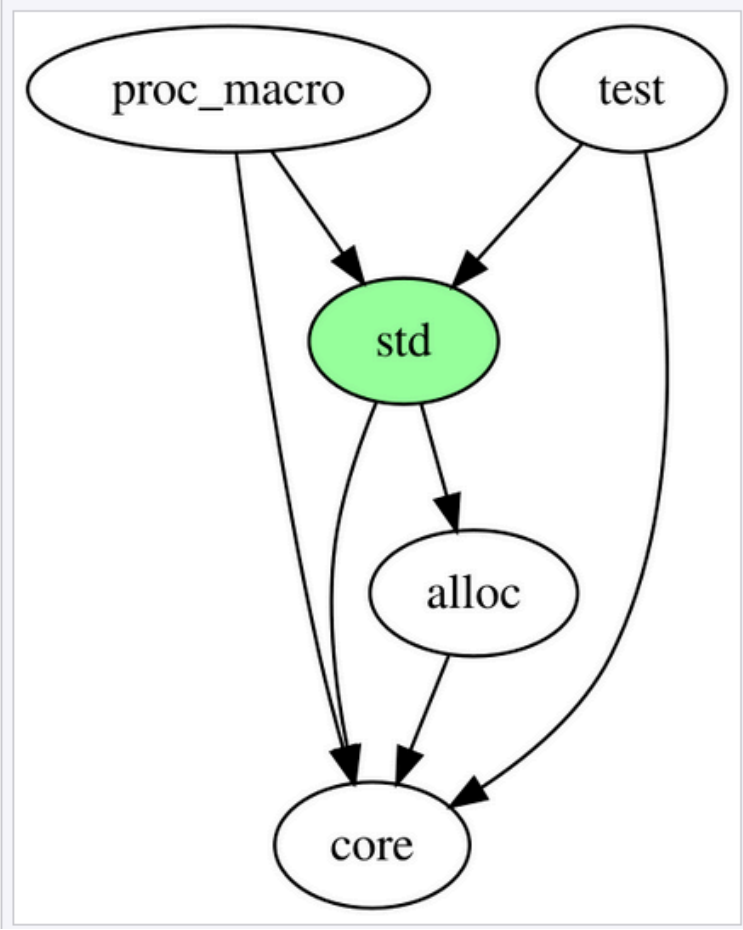
- Call the function from main and print value, or an error message in case of None.

Packages and Crates

- Crate is the smallest unit, binary or library.
- Binary crates start execution in main function.
- Library crates don't have a main function.
- Library crates share functionality with multiple projects.
- A package bundles multiple crates and contains a Cargo.toml
- A package contains at most one library.

Packages

std vs nostd



std vs nostd

Target	nostd Packages	std Packages
Embedded	core::* serde::* embassy::* ..	
Linux/Win/Mac		core::* std::* serde::* tokio::* ..

Creating a package

```
$ cargo new my-project --bin
    Creating binary (application) `my-project` package
note: see more `Cargo.toml` keys and their definitions at https
```

Creating the file structure

```
my-project/
├── Cargo.toml
└── src
    └── main.rs
```

Organization - Crate root

Start from the crate root src/main.rs or src/lib.rs.

```
my-project/
├── Cargo.toml
└── src
    └── main.rs
```

Organization - Modules

Declaring modules inline.

```
1
2  mod mytool {
3      // Code within a module is private by default
4      pub fn myfunc()   {   /* snip */ }
5  }
6  use mytool::myfunc; // create shortcut
7
8  fn main() {
9       println!("Hello, world!");
10      myfunc();
11 }
```

Organization - Modules

Declaring modules as file.

```

mod mytool;
use mytool::myfunc; // create shortcut

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

```
my-project/
├── Cargo.toml
└── src
    ├── main.rs
    └── mytool.rs
```

my-project/src/mytool.rs

```
// Code within a module is private by def
pub fn myfunc()  {  /* snip */ }
```

Organization - Modules

Declaring modules as directory.

```

mod mytool;
use mytool::myfunc; // create shortcut
fn main() {
    println!("Hello, world!");
    myfunc();
}
```

```
my-project/
├── Cargo.toml
└── src
    ├── main.rs
    └── mytool
        └── mod.rs
```

my-project/src/mytool/mod.rs

```
// Code within a module is private by def
pub fn myfunc()  {  /* snip */ }
```

Organization - Submodules

Inline

```
1
2 mod mytool {
3     // snip
4     mod network { /* snip */ }
5 }
6 fn main() {
7     println!("Hello, world!");
8     mytool::myfunc();
9 }
```

File-based

```
1 my-project/
2 ├── Cargo.toml
3 └── src
4     ├── main.rs
5     └── mytool
6         ├── mod.rs
7         └── network.rs
```

Directory-based

```
1 my-project/
2 ├── Cargo.toml
3 └── src
4     ├── main.rs
5     └── mytool
6         ├── mod.rs
7         └── network
8             └── mod.rs
```

Organization - Item Paths

- Absolute paths from crate root, eg

```
1 use crate::mytool::*; // with glob operator *
```

- Relative paths starting from current module, using 'self::', 'super::', or local identifier, eg

```
1 use self::mytool::*;
2 pub use super::mytool::myfunc; // re-export with 'pub use'
3 use std::collection::HashMap as StdHashMap; // re-name
4 use std::io;
5 use rand::Rng; // shortcut to Item in external package
6 // snip
```

Organization - Nested Paths

List consuming vertical space

```
1 // --snip--
2 use std::cmp::Ordering;
3 use std::io;
4 // --snip--
```

Instead use nested paths

```
1 // --snip--
2 use std::{cmp::Ordering, io};
3 // --snip--
```

Organization - Nested Paths

List consuming vertical space

```
1 // --snip--
2 use std::io;
3 use std::io::Write;
4 // --snip--
```

Instead use nested path with `self`.

```
1 // --snip--
2 use std::io::{self, Write};
3 // --snip--
```

Download from

https://github.com/frehberg/rust-demo-project

Install cargo workspace plugin

```
$ cargo install cargo-workspaces
```

Create Workspace

```
$ mkdir project
$ cargo workspaces init project
```

Create Members

```
$ cd rust-schulung
$ cargo workspaces create myapp  --bin
$ mkdir myapp/tests
$ cargo workspaces create mylib1 --lib
$ mkdir mylib1/tests
$ touch mylib1/build.rs
...
# repeat for member lecture02-types
```

Workspace

```
project/
├── Cargo.lock
├── Cargo.toml
├── .git
├── myapp
│   ├── Cargo.toml
│   └── src
│       └── main.rs
└── mylib1
    ├── build.rs
    ├── Cargo.toml
    ├── src
    │   ├── adapter.c
    │   └── lib.rs
    └── tests
```

Workspace Cargo Config

```
project/
├── Cargo.toml
├── .git
├── myapp
...
```

project/Cargo.toml - members & shared dependencies

```toml
[workspace]
resolver = "3"
members = [
    "myapp",
    "mylib1",]
[workspace.dependencies]
rand = "0.9"
cc = "1.0"
regex = { version = "1.11",
          default-features = false,
          features = ["std"] }
```

Dependency resolver 3

Recommended for Rust edition 2024

```
[workspace]
resolver = "3" # for Rust edition 2024
members = [
...
```

App Cargo Config

```
├── myapp
│   ├── Cargo.toml
│   └── src
│       └── main.rs
```

project/myapp/Cargo.toml defines dependecy to member mylib1

```
[package]
name = "myapp"
version = "0.0.0"
edition = "2024"

[dependencies]
mylib1    = { path = "../mylib1" }
```

Lib Cargo Config

```
└── mylib1
    ├── build.rs
    ├── Cargo.toml
    ├── src
    │   ├── adapter.c
    │   └── lib.rs
    └── tests
```

project/mylib1/Cargo.toml

```toml
[package]
name = "mylib1"
version = "0.0.0"
edition = "2024"

[dependencies]
regex = { workspace = true,
          features = ["std"] }

[build-dependencies]
cc.workspace = true
```

Lib Build-script

```
└── mylib1
    ├── build.rs
    ├── Cargo.toml
    ├── src
    │   ├── adapter.c
    │   └── lib.rs
    └── tests
```

project/mylib1/Cargo.toml

```
// Example custom build script.
fn main() {
    // Tell Cargo that if the given file changes, to rerun this b
    println!("cargo::rerun-if-changed=src/adapter.c");
    // Use the `cc` crate to build a C file and statically link i
    cc::Build::new()
        .file("src/adapter.c")
        .compile("adapter");
```

Lib Build-script

```
└── mylib1
    ├── build.rs
    ├── Cargo.toml
    ├── src
    │   ├── adapter.c
    │   └── lib.rs
    └── tests
```

project/mylib1/src/adapter.c

```c
static int INTERNAL_OBJ = 4711;

///
///
int*
get_singleton(char* object_id) {
  if (!strcmp("SINGLETON_A", object_id)) {
    return &INTERNAL_OBJ;
  }
  return NULL;
}
```

Rust Build-script

```
└── mylib1
    ├── build.rs
    ├── Cargo.toml
    ├── src
    │   ├── adapter.c
    │   └── lib.rs
    └── tests
```

project/mylib1/Cargo.toml

```rust
// Example custom build script.
fn main() {
    // Tell Cargo that if the given file changes, to rerun this b
    println!("cargo::rerun-if-changed=src/adapter.c");
    // Use the `cc` crate to build a C file and statically link i
    cc::Build::new()
        .file("src/adapter.c")
        .compile("adapter");
```

Hands On 2

- Create workspace with Cargo.
- Use the std::fs::File::open function to open a file and read content into String.
- Call the function from main and size of buffer.

Challenge: File-IO, Result-Enum, Std-IO-Error handling.

Hands On 3

- Define a String and convert into u8 byte-array
- Iterate the byte-array with map/filter and count the spaces.

Hands On 4

- Define a String (assume String is very large)
- Iterate the array with map/filter and 'take' first 10 capital letters.
- return the sequence of 10 capital letters

Collections

- Vector storing N values in contiguous memory.
- String is a collection of characters.
- HashMap associates specific keys each with a value.

Vector - Create

```
1 // creating empty vector
2 let v: Vec<i32> = Vec::new();
3
4 // usage of vec macro
5 let v = vec![1, 2, 3];
6
7 // create vector with 7 elements
8 let v = vec![42; 7];
```

Vector - Modify

Vector v must be mutable.

```
1 let mut v = Vec::new();
2
3 v.push(5);
4 v.push(6);
5 v.push(7);
6 v.push(8);
```

Vector - Read

```rust
let v = vec![1, 2, 3, 4, 5];
let third: &i32 = &v[2];  // may panic if index out of bound
println!("The third element is {third}");

let third: Option<&i32> = v.get(2); // may return Option::None
                                    // if index out of bound
match third {
    Some(third) => println!("The third element is {third}"),
    None => println!("There is no third element."),
}
```

Vector - Iterator Chaining

Iterators combined with functions like map, filter, and sum.

```
let result: i64 = (1..=100)
        .filter(|x: &i32| -> bool { x % 3 == 0 })
        .map(|x| -> i64 {x as i64})
        .sum();
```

String Container

```
let s1 = String::from("Hello, ");
let s2 = String::from("world!");
let s3 = s1 + &s2; // note s1 has been moved here
                   // and can no longer be used
let first_letter: Option<char> = s3.chars().nth(0);
let s3_slice = &s3[0..5];
```

Associated Key-Value Pairs - HashMap

```
let mut scores = HashMap::new();

scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Yellow"), 50);

let upper_scores :Vec<String>=  scores.iter()
        .filter(|(key,value)| **value > 20  )
        .map(|(key, _) | key)
        .collect();
```

String Container

```
let s1 = String::from("Hello, ");
let s2 = String::from("world!");
let s3 = s1 + &s2; // note s1 has been moved here
                   // and can no longer be used
let first_letter: Option<char> = s3.chars().nth(0);
let s3_slice = &s3[0..5];
```

Error Handling

- Recoverable Errors.
- Unrecoverable Errors.

Unrecoverable Error

For example caused by

- Accessing an array past the end.
- Integer overflow.

Strategies to end the execution

- unwind - heavy process releasing resources.
- abort - immediate termination, OS releasing resources.

Cargo.toml parameter

```
1 [profile.release]
2 panic = 'abort'
```

Recoverable Errors

For example caused by bad file IO

```
use std::fs::File;
use std::io::Error;
use std::result::Result;

fn main() {
    let result: Result<std::fs::File, std::io::Error> =
            File::open("hello.txt");
    match result {
        Ok(fout) => { /* write to open file */ },
        Err(error) => {
            println!("Failed to open file {error}");
        }
    }
// snip
}
```

Generics

- Generic Data Types.
- Traits: Defining shared behavior.

Generic Lists

```
 1 fn largest(list: &[i32]) -> &i32 {
 2     let mut largest = &list[0];
 3     for item in list { if item > largest { largest = item; } }
 4     largest
 5 }
 6
 7 fn main() {
 8     let number_list = vec![34, 50, 25, 100, 65];
 9
10     let result = largest(&number_list);
11     println!("The largest number is {result}");
12 }
```

Generic Parameters

Op '>' requires T implements PartialOrd trait

```
1  fn largest<T>(list: &[T]) -> &T
2     where T: PartialOrd {
3      let mut largest = &list[0];
4      for item in list { if item > largest { largest = item; } }
5      largest
6  }
7
8  fn main() {
9      let number_list = vec![34, 50, 25, 100, 65];
10
11     let result = largest(&number_list);
12     println!("The largest number is {result}");
13 }
```

PartialOrd Trait

```
1
2  pub trait PartialOrd<Rhs = Self>: PartialEq<Rhs>
3  where
4      Rhs: ?Sized,
5  {
6      // Required method
7      fn partial_cmp(&self, other: &Rhs) -> Option<Ordering>
8
9      // Provided methods
10     fn lt(&self, other: &Rhs) -> bool { ... }
11     fn le(&self, other: &Rhs) -> bool { ... }
12     fn gt(&self, other: &Rhs) -> bool { ... }
13     fn ge(&self, other: &Rhs) -> bool { ... }
14 }
```

Impl Trait For Custom Type

Op '>' requires T implements PartialOrd trait

```
1  struct Num (isize);
2  impl PartialOrd for Num {
3      fn partial_cmp(&self, other: &Self) -> Option<Ordering> {
4          self.0.partial_cmp(&other.0)
5      }
6  }
7  impl PartialEq for Num {
8      fn eq(&self, other: &Self) -> bool {
9          self.0 == other.0
10     }
11 }
12 fn  main() {
13     let number_list = vec![Num(34), Num(50), Num(25), Num(100)
14     let result = largest(&number_list);
15     // snip
```

Exercise: Read File

```
1
2  use std::io::{Error, Read};
3  use std::path::Path;
4
5  fn  read_file(name: &str) -> Option<String>  {
6      let path : &Path = Path::new(name);
7      match  File::open(path) {
8          Ok(mut f) => {
9              let mut buffer = String::new();;
10             // read the whole file
11              f.read_to_string(&mut buffer);
12             Some(buffer)
13         }
14         Err(e) => {
15             println!("Failed to read file {e}");
```

THE END