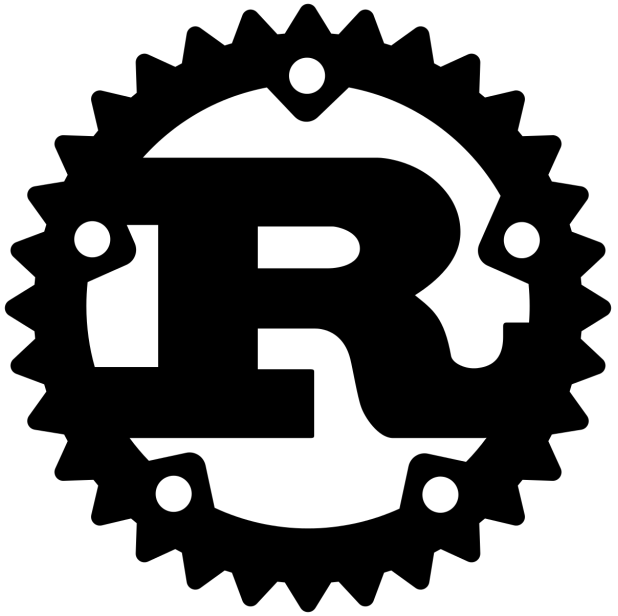


Voyage into uncharted lands s01e05

The Rust programming language

An introduction

Functional programming meetup • 2018-03-28 • Montpellier - France

About me

Baptiste

Back-end engineer

Teads (> 3 years)

@Viewtiful_

Montpellier community Slack: Viewtiful

A quality conference for tech enthusiasts

28 & 29 June

https://sunny-tech.io/

Tickets can be purchased & sponsoring is open!

History

2006 – Personal project of Graydon Hoare

2009 – Project sponsored by Mozilla

2010 – Official support by Mozilla

2011 – *rustc* successfully compiled itself after a re-write from Ocaml

2012 – First numbered pre-alpha release – *Servo* dev started

2015 – 1.0 – first stable release

Now & beyond – Stable release every 6 weeks through "*rolling release cycle*"

Steve Klabnik excellent talk on Rust history: https://www.youtube.com/watch?v=79PSagCD_AY

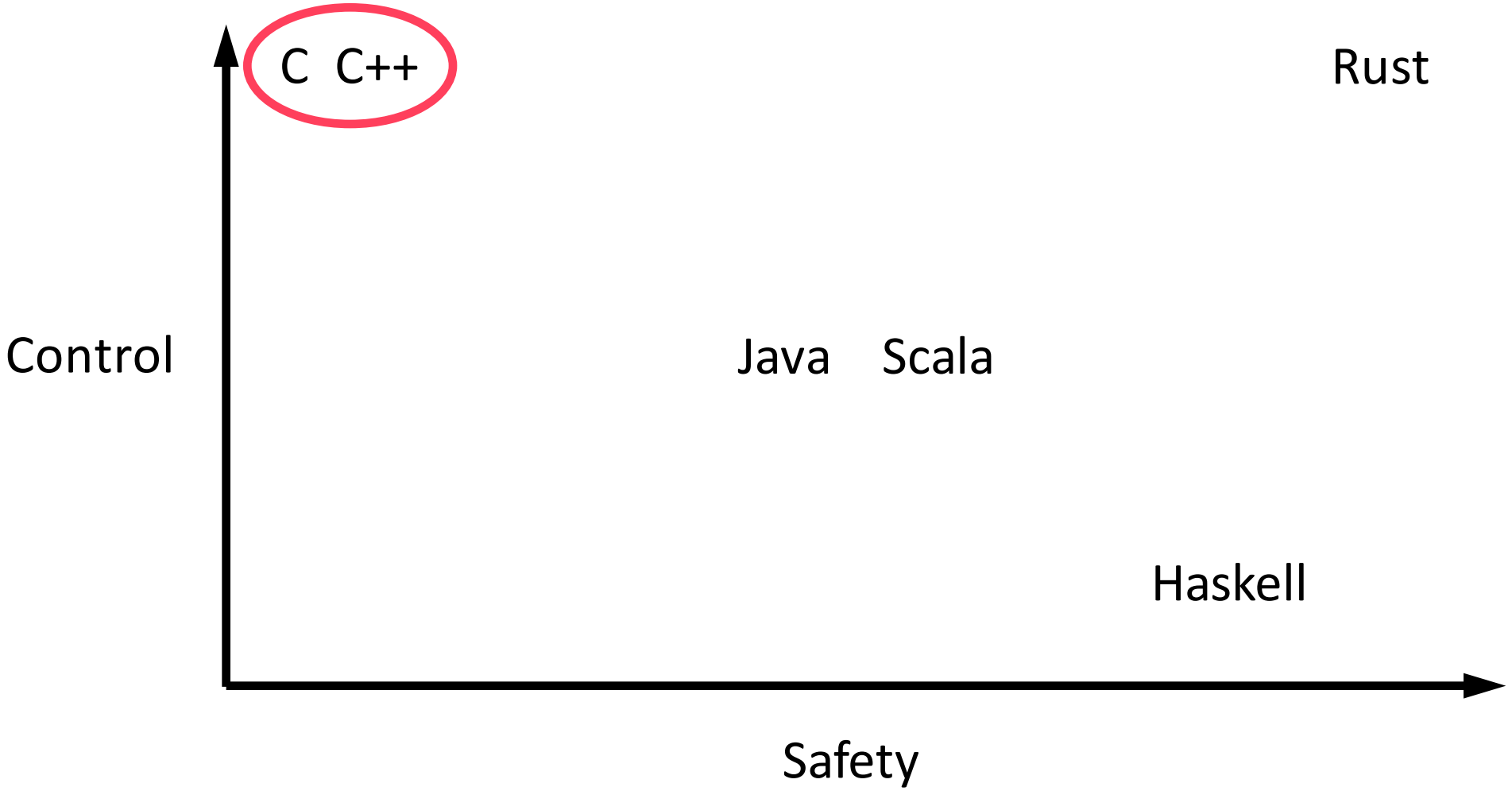
Motto

A systems *programming language* that runs blazingly fast, prevents segmentation fault, and guarantees thread safety.

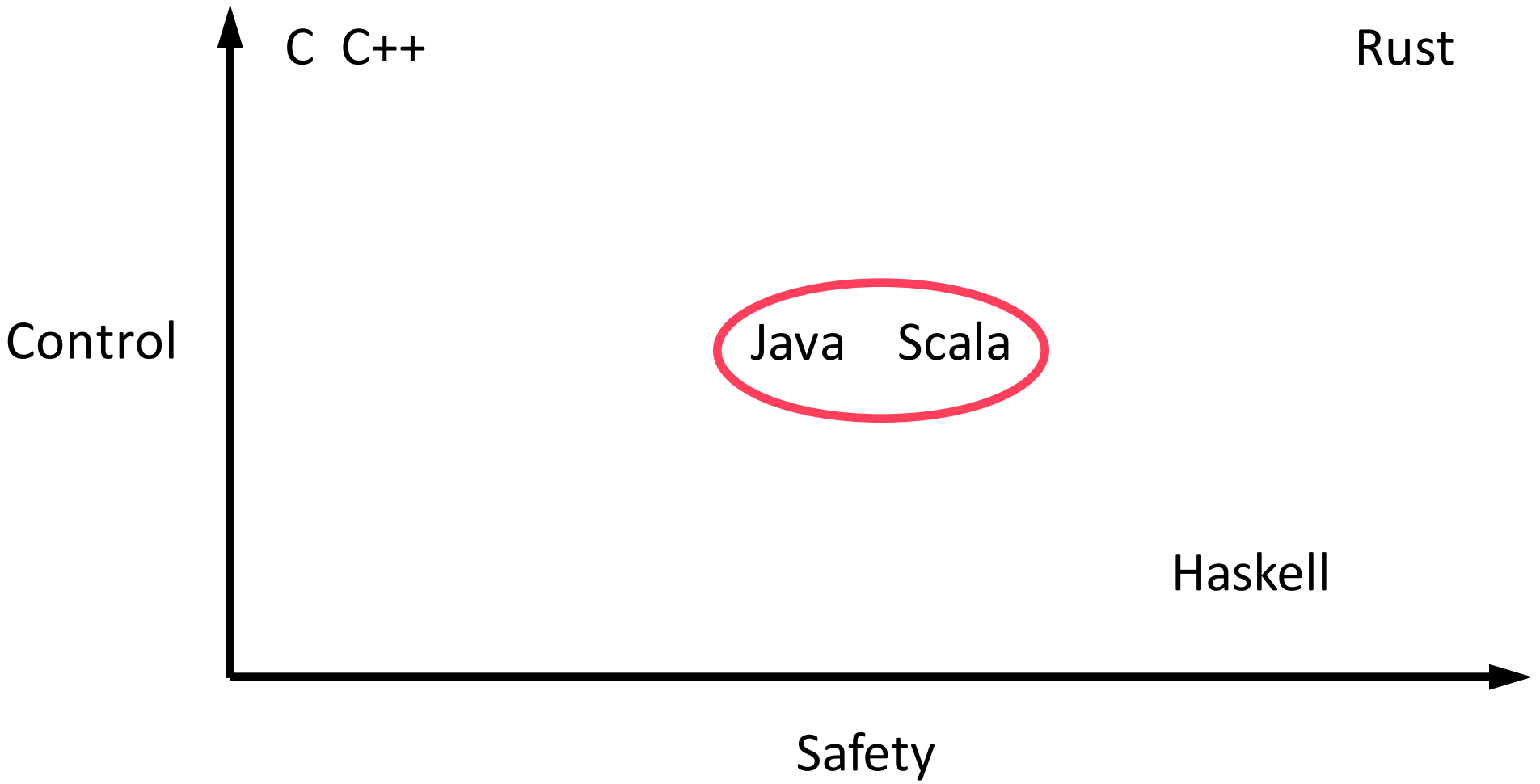
Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

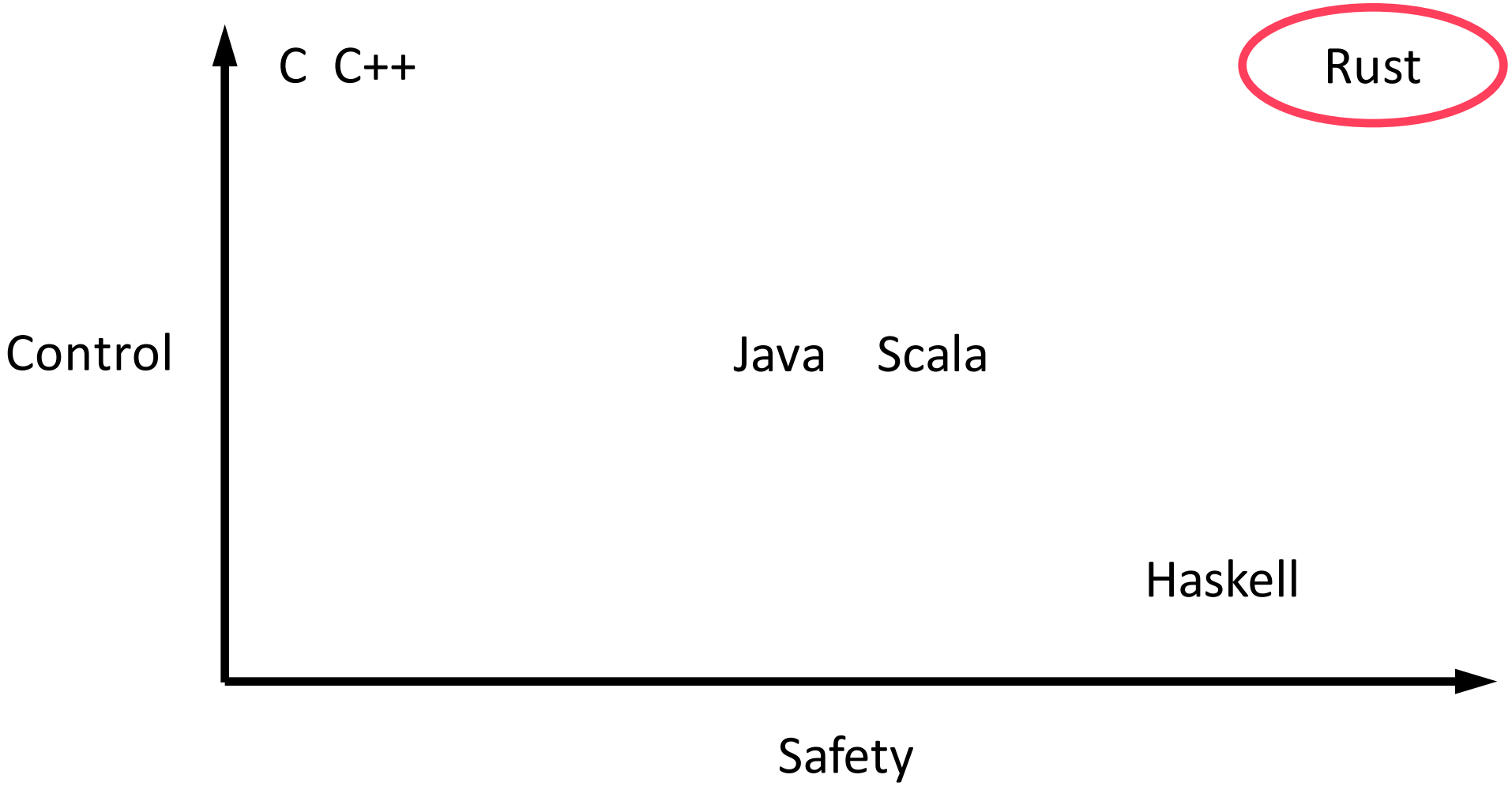
Rust vs. other languages



Rust vs. other languages



Rust vs. other languages



Rust killer features

Ownership

Borrowing

Lifetime

Borrow checker

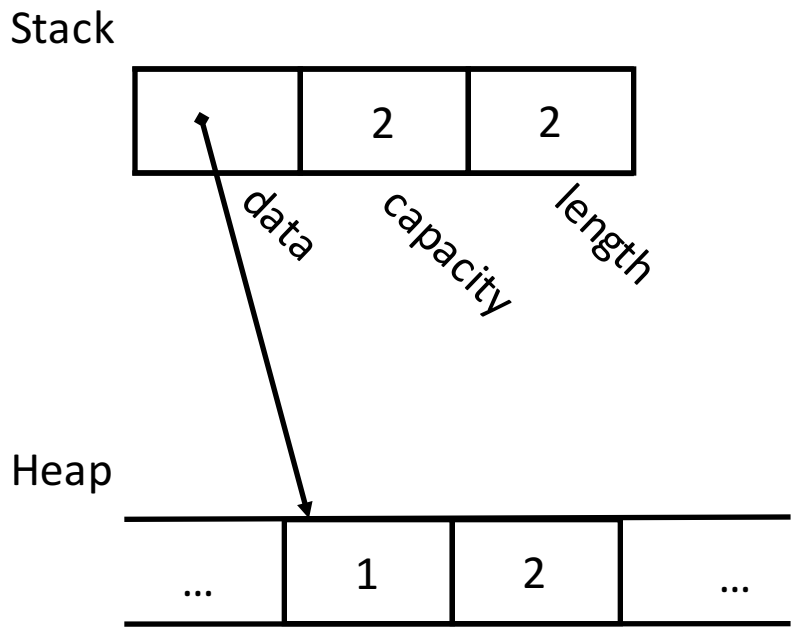
Ownership

- Every value has a single owner that determines its lifetime
- Owner is freed (goes out of scope), the value is also freed

```
1 fn main() {
2     let my_vec: Vec<i32> = vec![1, 2, 3]; // allocated here
3
4     println!("{:?}", my_vec);
5 }                                         // dropped here
```

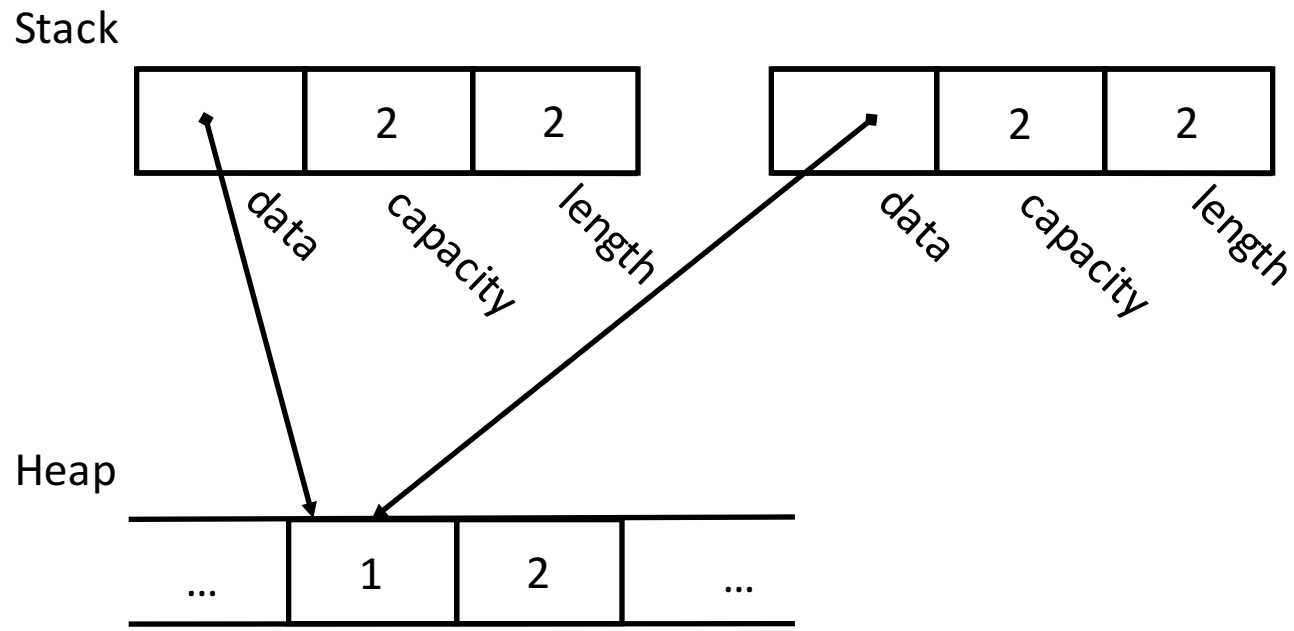
Ownership

```rust
fn print_vec(vec: Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(vec);

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



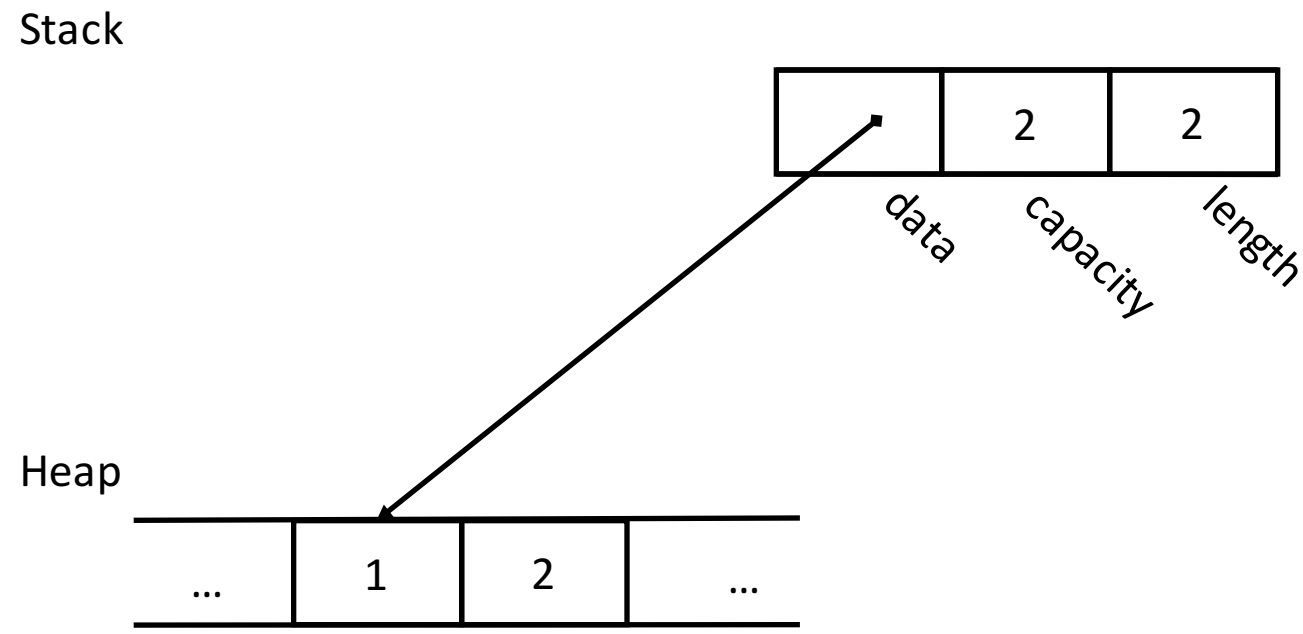
Ownership

```rust
fn print_vec(vec: Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(vec);

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



Ownership

```rust
fn print_vec(vec: Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(vec);

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



Ownership

```rust
fn print_vec(vec: Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(vec);

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


Stack

Heap

Ownership

```rust
fn print_vec(vec: Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(vec);

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

```
error[E0382]: use of moved value: `vec`
 --> src/main.rs:9:22
  |
7 |     print_vec(vec);  // print_vec take ownership of the vector
  |               --- value moved here
8 |
9 |     println!("{:?}", vec);
  |                       ^^^ value used here after move
  |
  = note: move occurs because `vec` has type `std::vec::Vec<i32>`, which does not implement the `Copy` trait
```

Rust compiler error index: https://doc.rust-lang.org/error-index.html

Ownership

To make it works you must pass the values back and forth

```rust
fn print_vec(vec: Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(vec);

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

```rust
fn print_vec(vec: Vec<i32>) -> Vec<i32> {
    println!("{:?}", vec);

    vec
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    let vec: Vec<i32> = print_vec(vec);

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

Ownership

Basic types are copied (extend the trait Copy)

```
1 fn print_int(i: i32) {
2     println!("{:?}", i);
3 }
4
5 fn main() {
6     let i = 1;
7     print_int(i);
8
9     println!("{:?}", i);
10 }
```

```
1
1
```

Ownership: Aliasing vs. Mutability

Prevents the aliasing but authorize the mutability of the value

- ✗ Aliasing - ✓ mutability

Borrowing

Would be cumbersome to only have the concept of ownership

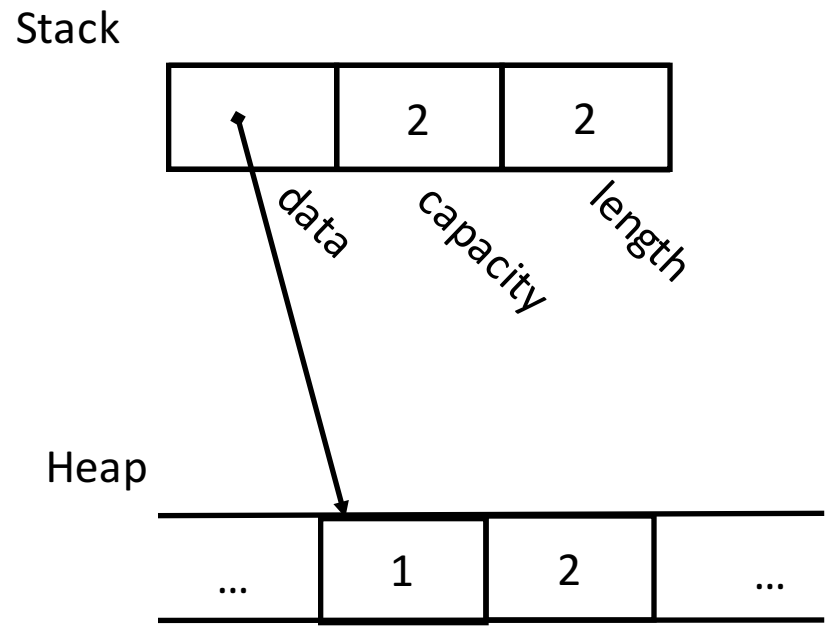
Concept of borrowing: Able to borrow the value for a short period of time

2 kinds of borrows:

- Shared reference (& T) – immutable references
  - ✓ Aliasing - ✗ mutability
- Mutable reference (&mut T)
  - ✗ Aliasing - ✓ mutability

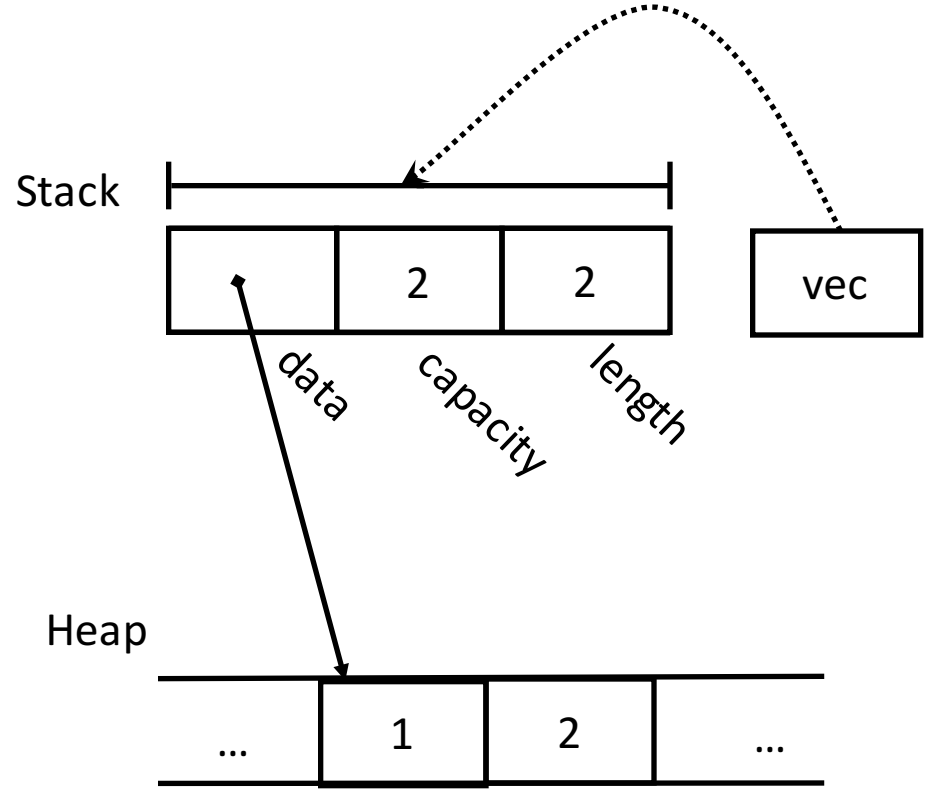
Shared reference (& T)

```rust
fn print_vec(vec: &Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(&vec);

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



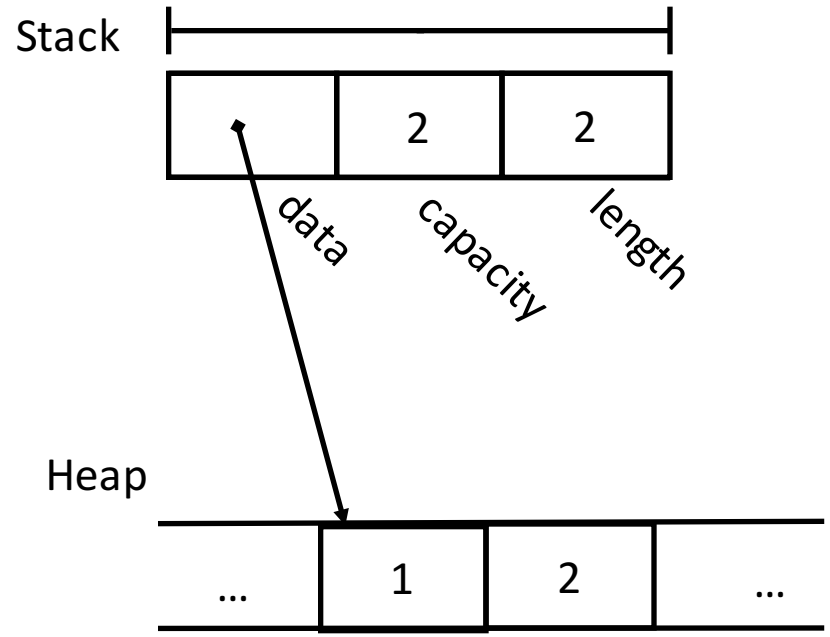
Shared reference (& T)

```rust
fn print_vec(vec: &Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(&vec);

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



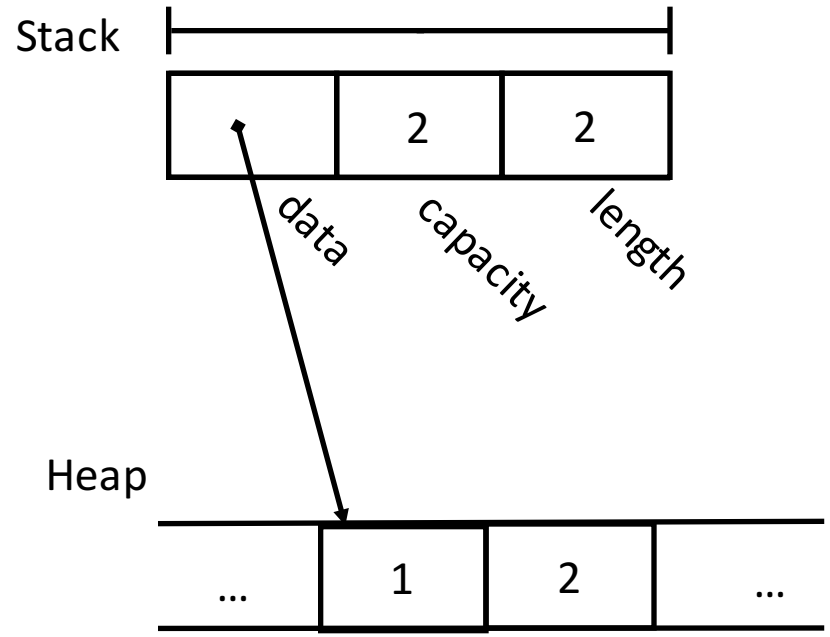
Shared reference (& T)

```rust
fn print_vec(vec: &Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(&vec);

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



Shared reference (& T)

```rust
fn print_vec(vec: &Vec<i32>) {
    println!("{:?}", vec);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(&vec);

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



Mutable reference (&mut T)

```rust
fn print_vec(vec: &mut Vec<i32>) -> () {
    println!("{:?}", vec);
    vec.push(3);
}

fn main() {
    let mut vec: Vec<i32> = Vec::new();
    vec.push(1);
    vec.push(2);

    print_vec(&mut vec);

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

```
[1, 2]
[1, 2, 3]
```

Lifetime <‘a>

Lifetimes are here to avoid dangling references

Lifetime: Scope for which the reference is valid

Used explicitly by the borrow checker

Lifetime elision: implicit lifetimes for the programmer

But, we sometimes need to be explicit to help the borrow checker understand what we are trying to achieve

Lifetime <‘a>

```rust
fn main() {
    let mut z;      // lifetime 'a

    {
        let y = 1;  // lifetime 'b
        z = &y;
    }

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

```
error[E0597]: `y` does not live long enough
  --> src/main.rs:6:14
   |
6  |         z = &y;
   |              ^ borrowed value does not live long enough
7  |     }
   |     - `y` dropped here while still borrowed
...
10 | }
   | - borrowed value needs to live until here
```

Lifetime <‘a>

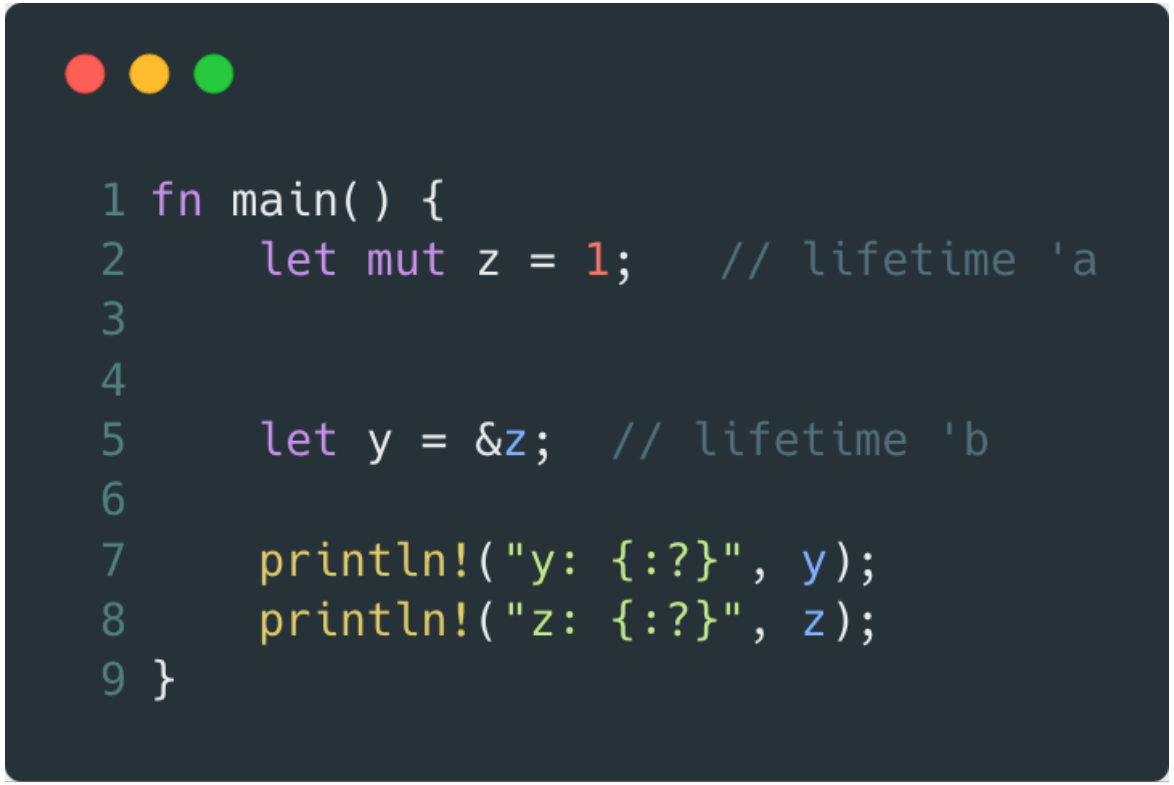
```
 1 fn main() {
 2     let mut z;   // lifetime 'a
 3
 4
 5     let y = 1;   // lifetime 'b
 6
 7     z = &y;
 8
 9     println!("{:?}", z);
10 }
11
```

Lifetime <‘a>

```rust
fn main() {
    let mut z;   // lifetime 'a

    let y = 1;   // lifetime 'b

    z = &y;

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

```
error[E0597]: `y` does not live long enough
  --> src/main.rs:6:10
   |
6  |     z = &y;
   |          ^ borrowed value does not live long enough
...
10 | }
   | - `y` dropped here while still borrowed
   |
   = note: values in a scope are dropped in the opposite order they are created
```

Lifetime <‘a>



```
1 fn main() {
2     let mut z = 1;    // lifetime 'a
3
4
5     let y = &z;  // lifetime 'b
6
7     println!("y: {:?}", y);
8     println!("z: {:?}", z);
9 }
```

Lifetime <‘a>

```rust
fn first_or_second(z: &i32, y: &i32) -> &i32 {
    if *z < 0 { y }
    else { z }
}


fn main() {
    let z = -1;   // lifetime 'a

    let y = 22;   // lifetime 'b

    println!("x: {:?}", first_or_second(&z, &y));
}
```

```
error[E0106]: missing lifetime specifier
 --> src/main.rs:1:41
  |
1 | fn first_or_second(z: &i32, y: &i32) -> &i32 {
  |                                         ^ expected lifetime parameter
  |
  = help: this function's return type contains a borrowed value, but the signature does not say whether it is borrowed from `z` or `y`
```

Lifetime <‘a>

```
1 fn first_or_second<'a>(z: &'a i32, y: &'a i32) -> &'a i32 {
2     if *z < 0 { y }
3     else { z }
4 }
5
6
7 fn main() {
8     let z = -1;   // lifetime 'a
9
10    let y = 22;   // lifetime 'b
11
12    println!("x: {:?}", first_or_second(&z, &y));
13 }
14
```

Ownership, borrowing, lifetimes

- Enable:
  - No need for runtime (check at compile time)
  - Memory safety without GC
  - No data-races

Unsafe Rust

Unsafe Rust

```
1 fn main() {
2     let mut a = &mut 5 as *mut i32;
3
4     println!("a is: {}", *a);
5 }
```

```
error[E0133]: dereference of raw pointer requires unsafe function or block
 --> src/main.rs:6:26
  |
6 |     println!("a is: {}", *a);
  |                          ^^ dereference of raw pointer
```

Unsafe Rust

```rust
fn main() {
    let mut a = &mut 5 as *mut i32;

    unsafe {
        println!("a is: {}", *a);
    }
}
```

```
a is: 5
```

Unsafe Rust

```rust
use std::ptr;

fn main() {
    let mut a = &mut 5 as *mut i32;

    unsafe {
        a = ptr::null_mut();
        println!("a is: {}", *a);
    }
}
```

```
5 Segmentation fault      timeout --signal=KILL ${timeout}
```

Is Rust a functional programming language?

Humble Book Bundle FUNCTIONAL PROGRAMMING By O'REILLY

Is Rust a functional programming language?

Not really,

Rust is a multi-paradigm language:

- imperative-procedural
- oriented object
- functional

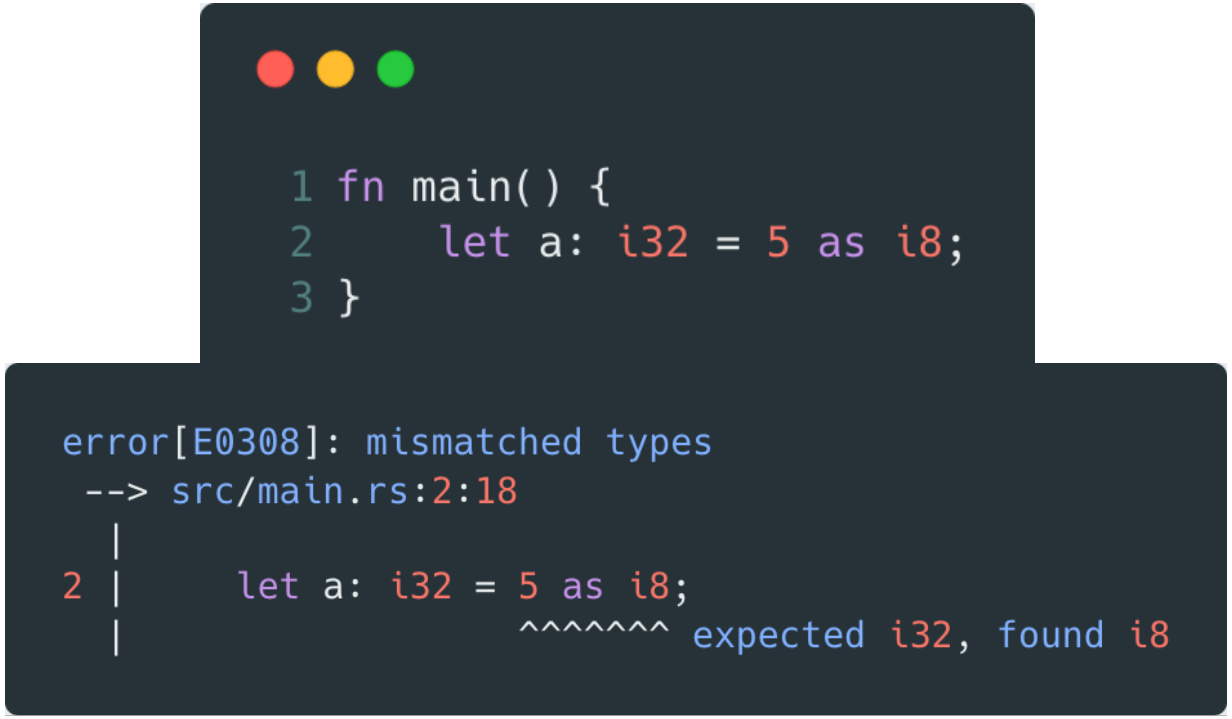
Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)

Is Rust a functional programming language?

Types do not change under the hood, expects type **X**, you need to give it type **X**



Is Rust a functional programming language?

Primitive types

array	A fixed-size array, denoted `[T; N]`, for the element type, `T`, and the non-negative compile-time constant size, `N`.
bool	The boolean type.
char	A character type.
f32	The 32-bit floating point type.
f64	The 64-bit floating point type.
fn	Function pointers, like `fn(usize) -> bool`.
i8	The 8-bit signed integer type.
i16	The 16-bit signed integer type.
i32	The 32-bit signed integer type.
i64	The 64-bit signed integer type.
isize	The pointer-sized signed integer type.
pointer	Raw, unsafe pointers, `*const T`, and `*mut T`.
reference	References, both shared and mutable.
slice	A dynamically-sized view into a contiguous sequence, `[T]`.
str	String slices.
tuple	A finite heterogeneous sequence, `(T, U, ..)`.
u8	The 8-bit unsigned integer type.
u16	The 16-bit unsigned integer type.
u32	The 32-bit unsigned integer type.
u64	The 64-bit unsigned integer type.
unit	The `()` type, sometimes called "unit" or "nil".
usize	The pointer-sized unsigned integer type.
i128	[Experimental] The 128-bit signed integer type.
never	[Experimental] The `!` type, also called "never".
u128	[Experimental] The 128-bit unsigned integer type.

Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)

Is Rust a functional programming language?

Immutability by default

```rust
fn main() {
    let a: i32 = 5;

    a = 3;
}
```

```
error[E0384]: cannot assign twice to immutable variable `a`
 --> src/main.rs:4:5
  |
2 |     let a: i32 = 5;
  |         - first assignment to `a`
3 |
4 |     a = 3;
  |     ^^^^^ cannot assign twice to immutable variable
```

Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)
- Traits are like *typeclasses*

Is Rust a functional programming language?

Traits: Defining shared behavior

```rust
pub trait Printable {
    fn printable(&self) -> ();
}

impl Printable for i32 {
    fn printable(&self) -> () { println!("{:?}", self); }
}

fn main() {
    let a: i32 = 4;

    a.printable();
}
```

Is Rust a functional programming language?

Traits: Trait bound

```rust
pub trait Printable {
    fn printable(&self) -> ();
}

impl Printable for i32 {
    fn printable(&self) -> () { println!("{:?}", self); }
}

fn need_printable<T: Printable>(value: T) -> () {
    value.printable();
}

fn main() {
    let a: i32 = 4;

    need_printable(a);
}
```

Is Rust a functional programming language?

Traits: Trait bound

```rust
pub trait Printable {
    fn printable(&self) -> ();
}

impl Printable for i32 {
    fn printable(&self) -> () { println!("{:?}", self); }
}

fn need_printable<T: Printable>(value: T) -> () {
    value.printable();
}

fn main() {
    let a: i32 = 4;
    let b: i8 = 3;

    need_printable(a);
    need_printable(b);
}
```

```
error[E0277]: the trait bound `i8: Printable` is not satisfied
  --> src/main.rs:18:5
   |
18 |     need_printable(b);
   |     ^^^^^^^^^^^^^^^ the trait `Printable` is not implemented for `i8`
   |
   = note: required by `need_printable`
```

Is Rust a functional programming language?

Traits: multiple trait bound

```
1 fn need_printable_and_display<T: Printable + Display>(value: T) -> () {
2     value.printable();
3 }
```

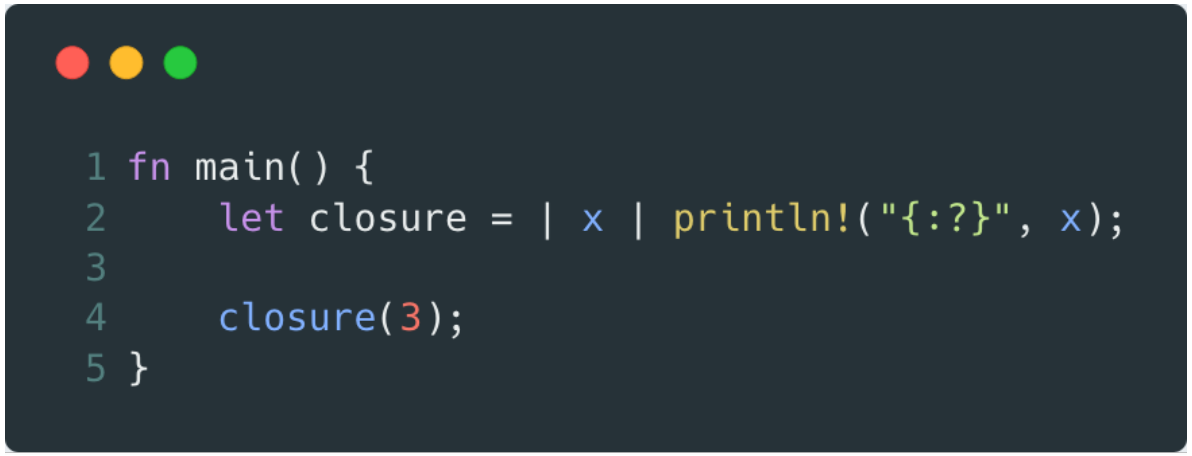
Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)
- Traits are like *typeclasses*
- Closures

Is Rust a functional programming language?

Closures



```rust
fn main() {
    let closure = | x | println!("{:?}", x);

    closure(3);
}
```

Is Rust a functional programming language?

Closures: type inference

```rust
fn main() {
    let closure = | x | println!("{:?}", x);

    closure(3);
    closure("Hey!");
}
```

```
error[E0308]: mismatched types
 --> src/main.rs:5:13
  |
5 |     closure("Hey!");
  |             ^^^^^^ expected integral variable, found reference
  |
  = note: expected type `{integer}`
             found type `&'static str`
  = help: here are some functions which might fulfill your needs:
          - .len()
```

Is Rust a functional programming language?

Closures: implement one of the following trait

- Fn : borrowing immutably variables captured
- FnMut: borrowing mutably variables captured
- FnOnce : Take ownership of the variables it captured from its enclosing scope

Is Rust a functional programming language?

FnOnce

```rust
fn main() {
    let vec: Vec<i32> = vec![1];
    let closure = move | x: i32 | {
        println!("x: {:?}, vec: {:?}", x, vec);
    };

    closure(1);

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

```
error[E0382]: use of moved value: `vec`
 --> src/main.rs:9:27
  |
3 |     let closure = move | x: i32 | {
  |                   -------------- value moved (into closure) here
...
9 |     println!("vec: {:?}", vec);
  |                           ^^^ value used here after move
  |
  = note: move occurs because `vec` has type `std::vec::Vec<i32>`, which does not implement the `Copy` trait
```

Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)
- Traits are like *typeclasses*
- Closures
- Iterators (lazy, higher order functions – map, filter, fold, collect, …)

Is Rust a functional programming language?

Iterators

```rust
fn main() {
    let vec = (1..)
        .filter(|x| x % 2 == 0)
        .take(3)
        .map(|x| x / 2)
        .collect::<Vec<i32>>();

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

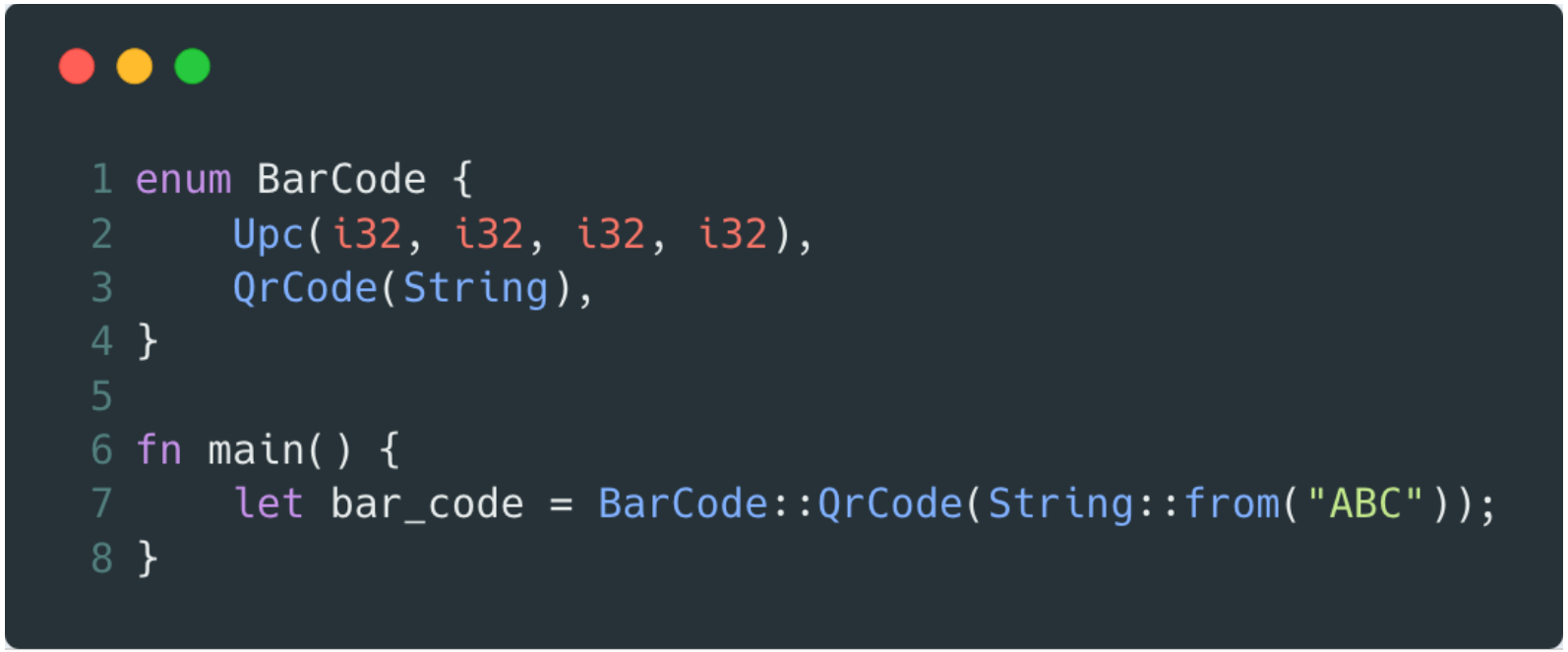
Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)
- Traits are like *typeclasses*
- Closures
- Iterators (lazy, support higher order functions – map, filter, fold, …)
- Enumeration are like *algebraic data types*

Is Rust a functional programming language?

Enumeration are like *algebraic data types*



```rust
enum BarCode {
    Upc(i32, i32, i32, i32),
    QrCode(String),
}

fn main() {
    let bar_code = BarCode::QrCode(String::from("ABC"));
}
```

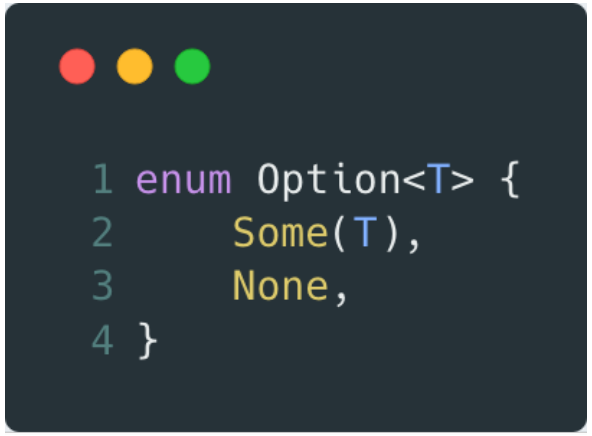
Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)
- Traits are like *typeclasses*
- Closures
- Iterators (lazy, support higher order functions – map, filter, fold, …)
- Enumeration are like *algebraic data types*
- Optional

Is Rust a functional programming language?

Optional



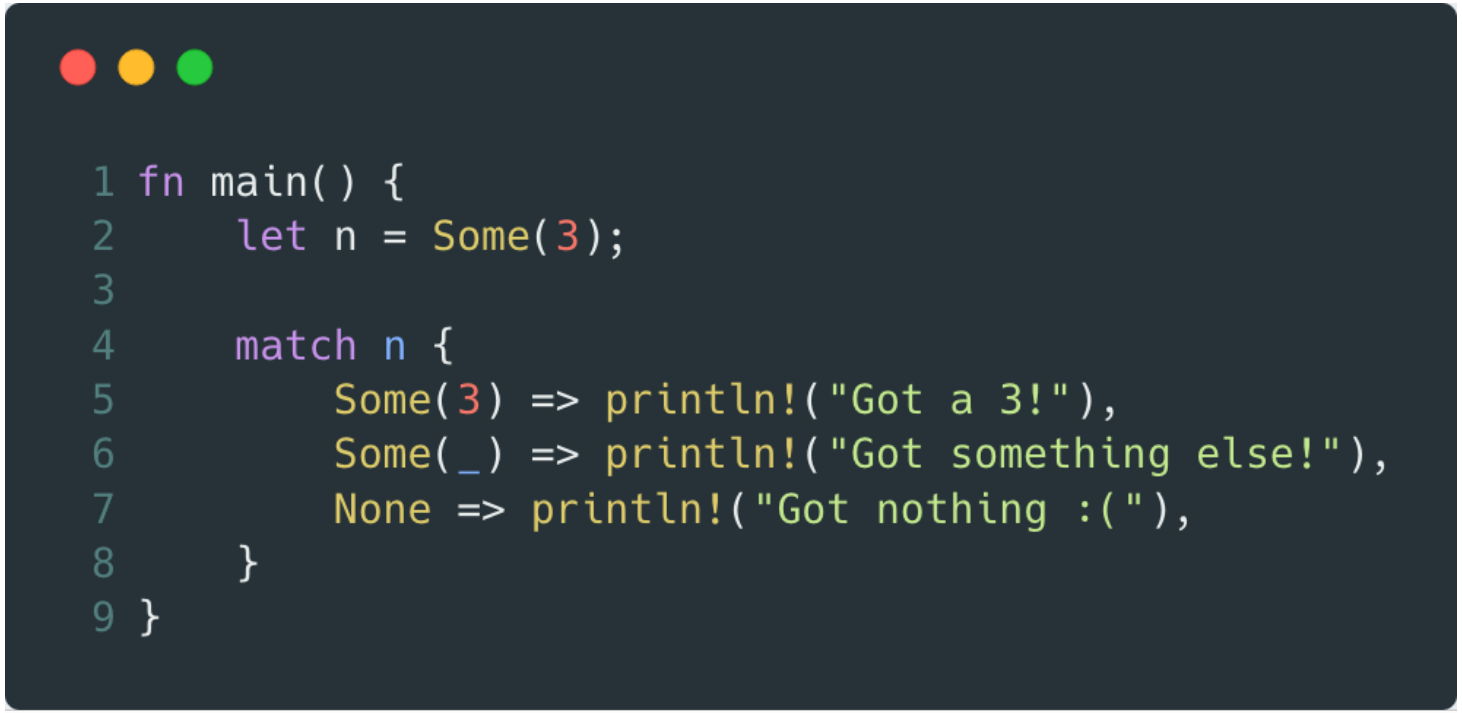
Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)
- Traits are like *typeclasses*
- Closures
- Iterators (lazy, support higher order functions – map, filter, fold, …)
- Enumeration are like *algebraic data types*
- Optional
- Pattern matching

Is Rust a functional programming language?

Pattern matching



```rust
fn main() {
    let n = Some(3);

    match n {
        Some(3) => println!("Got a 3!"),
        Some(_) => println!("Got something else!"),
        None => println!("Got nothing :("),
    }
}
```

Is Rust a functional programming language?

Take certain aspect from the functional paradigm

- Strong type system (affine and region types | statically typed | types do not change under the hood, expects type **X**, you need to give it type **X** | Lots of types in the STD | no exceptions)
- Emphasis on immutability but most of the time you use types in mutable way (which is safe, thanks to the borrow checker)
- Traits are like *typeclasses*
- Closures
- Iterators (lazy, support higher order functions – map, filter, fold, …)
- Enumeration are like *algebraic data types*
- Optional
- Pattern matching

These features are useful but are nothing fancy from a functional programmer perspective

Is Rust a functional programming language?

But is not pure

- Concept of purity (by default!) existed but was removed around 2011
  - Part of the reason: guarantees that it's impossible to access shared mutable state (read & write)

No *higher kinded types* but *associated-type constructors / families* are present!

- RFC: https://github.com/rust-lang/rfcs/blob/master/text/1598-generic_associated_types.md
- RFC PR: https://github.com/rust-lang/rfcs/pull/1598
- Podcast going through it: https://request-for-explanation.github.io/podcast/ep4-literally-haskell/index.html
- Article explaining the concept: http://smallcultfollowing.com/babysteps/blog/2016/11/02/associated-type-constructors-part-1-basic-concepts-and-introduction/

RFCs

- Important changes are discussed via RFCs
  - Design explanation
  - Consensus among the Rust community and the sub-teams
  - Github repository: https://github.com/rust-lang/rfcs
  - Example: “Add unstable sort to libcore” (https://github.com/rust-lang/rfcs/pull/1884)

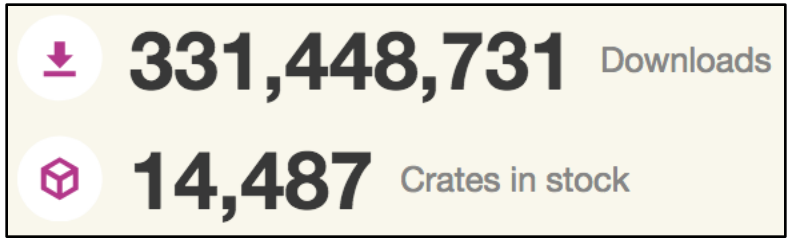
Cargo

Rust package manager

Cargo

- Project creation
- Handle Rust project's dependencies
- Compilation
- Testing
- Benchmarking
- Packaging
- Upload to crate.io (https://crates.io)
- Conventions to make working with Rust projects easier (RustFmt, …)
- Semantic versioning (https://semver.org/)
- Documentation: https://doc.rust-lang.org/cargo/index.html

- **Goal**: Ensure that you will always have a repeatable build



Rust nursery

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

Rust installation

Now: **Rustup** ! (https://www.rustup.rs/)

```
$ curl https://sh.rustup.rs -sSf | sh
```

- Install rust, cargo, documentation of the standard library
- Guide: https://github.com/rust-lang-nursery/rustup.rs
- Release channel: Stable, Beta, Nightly
- Update rustup: "*rustup self update*"
- New version of Rust? "*rustup update*"
- Install nightly: "*rustup install nightly*"
- Switch to nightly: "*rustup default nightly*"
- …

RustFmt

- Available since Rust > 1.24
- Github code: https://github.com/rust-lang-nursery/rustfmt
- RFCs for defining the style: https://github.com/rust-lang-nursery/fmt-rfcs

Rust-clippy

- A collection of lints to catch common mistakes and improve your Rust code
- 208 lints
- Repository: https://github.com/rust-lang-nursery/rust-clippy

Rust-wasm - Rust + WebAssembly = ❤

- Focus for 2018
- Interoperating JS and Rust code: https://github.com/alexcrichton/wasm-bindgen
- Book: https://rust-lang-nursery.github.io/rust-wasm/
- Repository: https://github.com/rust-lang-nursery/rust-wasm

STD - Rust Standard Library documentation

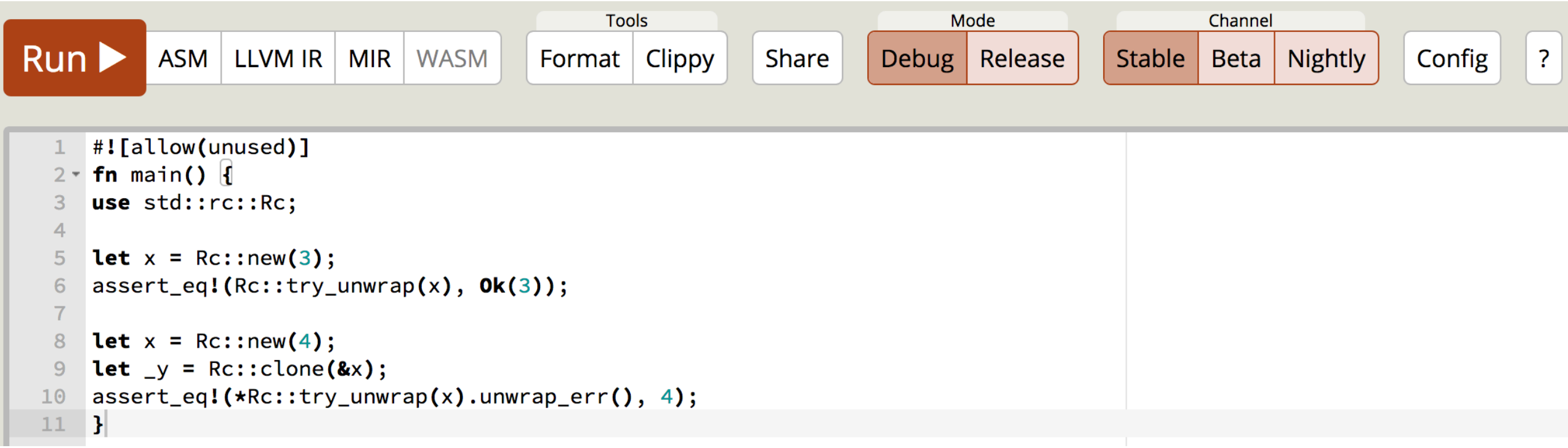
- Written inside the code source
- Versioned
- Descriptions with runnable examples
- Example
  - println! macro: https://doc.rust-lang.org/stable/std/macro.println.html
  - std::Rc: https://doc.rust-lang.org/stable/std/rc/index.html

IDE / Editor

- Racer (first commit: 4 March 2014): auto-completion tool
  - Repository: https://github.com/racer-rust/racer
- Rust Language Server (first commit: 31 August 2016) :
  - Runs in the background
  - Designed to be frontend-independent
  - Provides information about Rust programs
  - Uses the compiler and Racer
  - RFC: https://github.com/rust-lang/rfcs/pull/1317
  - Repository: https://github.com/rust-lang-nursery/rls
- IDE plugins
  - Eclipse
  - IntelliJ IDEA
  - Visual Studio
- Editor plugins
  - Emacs
  - Vim
  - Sublime Text
  - Atom
  - Visual Studio Code

IDE support tracklist: https://areweideyet.com/

Rust playground (https://play.rust-lang.org/)



Crates

Crates – Lots of them!

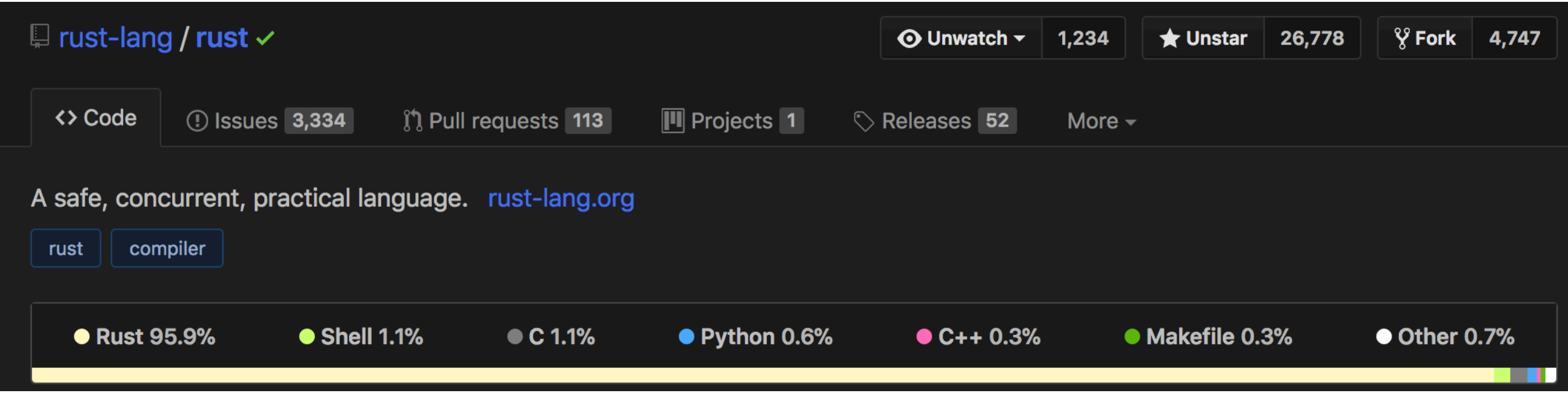
- Astronomy
- Audio
- Bioinformatics
- Caching
- Cloud
- Command line tools
- Compression
- Computation
- Cryptography
- Data processing
- Distributed systems
- Game development
- Graphics
- GUI
- Image processing
- Logging
- Mobile
- Network
- Parser
- Scripting
- Web programming
- Text processing
- Database
- HTTP client/server
- Encoding
- Asynchronous / Concurrency
- …

Curated list: https://github.com/rust-unofficial/awesome-rust

Who is using Rust?

Who is using Rust?

- Rust itself!



Who is using Rust?

Servo (https://servo.org/

Parts of Firefox:

- MP4 metadata parser
- Stylo: a pure-Rust parallel CSS engine introduced in Quantum

npm

Dropbox

CoreOS

clever cloud

...

https://www.rust-lang.org/en-US/friends.html

Software Engineer

Apple ★★★★☆ 4,865 reviews - Santa Clara Valley, CA

Apply Now

Come help us build the next generation cloud platform to support Internet services across Apple. Our Datacenter & Infrastructure Runtime team evolves, designs and deploys and operates infrastructure which forms the foundation for some of our most exciting services, including Siri, iCloud, Maps, iTunes, and more. The scale at which Apple operates requires the highest levels of automation and integration from the application through compute and the network, end to end. The strongest candidates will have both solid Linux systems expertise and proven software development skills. In this role you will have the unique opportunity to participate in delivering some of the world's largest-scale cloud services to the most-loved devices on Earth.

Key Qualifications

- 5 or more years in systems engineering and operations.
- Extensive experience in configuration management and fleet orchestration via Puppet, Chef, Ansible, or others.
- Fundamental understanding of distributed systems including: the CAP Theorem, microservices, and the Twelve Factor App.
- Deep understanding of the Linux operating system, including kernel, memory, process, threads, cgroups, static / shared libraries, IPC, signals.
- Comfort with extremely large fleets, each composed of tens of thousands of containers.
- Demonstrated history in automating operations processes via services and tools.
- Fluency in one or more high-level programming languages like Python, Go, Ruby, or Rust.
- Mentor and lead-level troubleshooting skills and methods.
- Passion for learning new abilities in cross-functional environments.

Software Engineer, Source Control

Facebook ★★★★☆ 260 reviews - Menlo Park, CA

Apply Now

(Menlo Park, CA)
Facebook's mission is to give people the power to build community and bring the world closer together. Through our family of apps and services, we're building a different kind of company that connects billions of people around the world, gives them ways to share what matters most to them, and helps bring people closer together. Whether we're creating new products or helping a small business expand its reach, people at Facebook are builders at heart. Our global teams are constantly iterating, solving problems, and working together to empower people around the world to build community and connect in meaningful ways. Together, we can help people build stronger communities – we're just getting started.

Facebook is seeking an experienced Software Engineer to join the Source Control team. A love of data structures and algorithms is a necessity. The Source Control team is building the next generation, distributed source control system designed to improve collaboration, support extremely large monolithic repositories, and have sub-100ms latency for all commands. We are looking for candidates who can build tools and platforms that improve the developer experience while scaling through multiple orders of magnitude. This position is full-time and is based in our main office in Menlo Park, CA.

Responsibilities

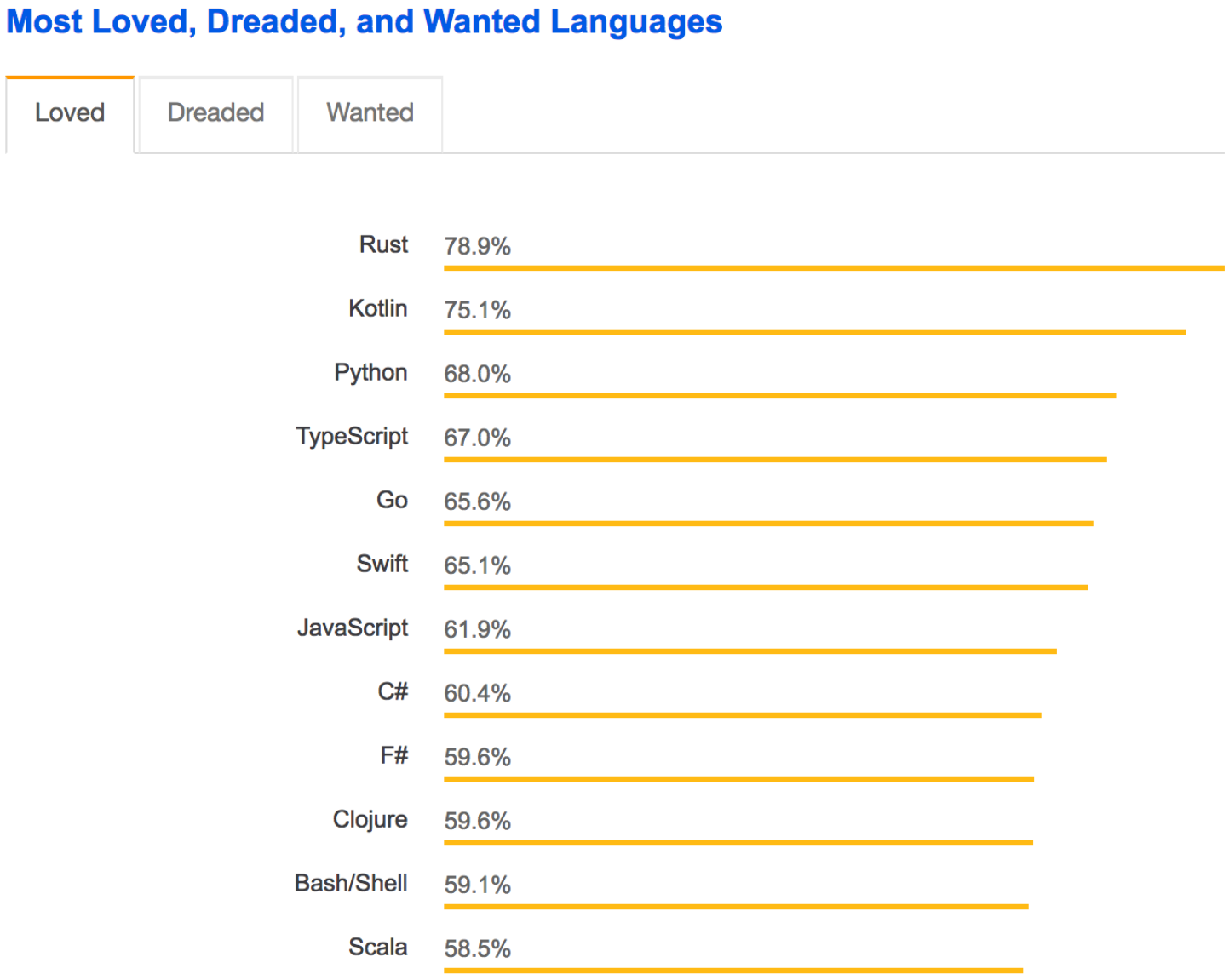
- Advance the state-of-the-art of source control systems through better algorithms, improved wire protocols, and novel user experiences
- Code using primarily Rust, Python, and C++ Interface with other teams to incorporate their innovations and vice versa
- Conduct design and code reviews
- Analyze and improve efficiency, scalability, and stability of various systems

Minimum Qualifications

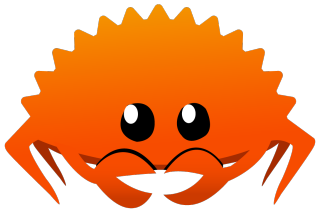
- 6+ years coding experience in C++ and Python
- Experience with Rust

Most loved language according to Stack Overflow

3 years in a row!!!

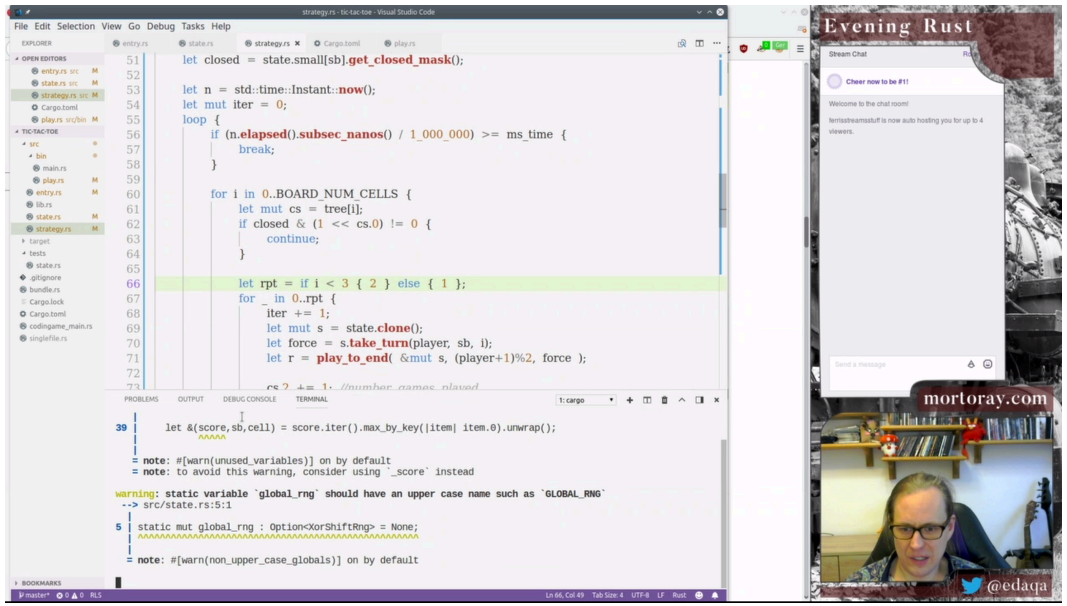


Community (https://www.rust-lang.org/en-US/community.html)



- IRC channels – irc.mozilla.org, #rust, #rust-beginners
- https://www.reddit.com/r/rust
- Forum: https://users.rust-lang.org/
- Finding *Rustaceans*: https://www.rustaceans.org/

Community – Live streaming



https://www.twitch.tv/ferrisstreamsstuff

https://www.twitch.tv/mortoray

Twitch Rust community: https://www.twitch.tv/communities/rustlang

This Week in Rust

- *Handpicked Rust updates, delivered to your inbox.*
- Weekly newsletter
- Subscription page: https://this-week-in-rust.org/
- Jobs offering

Read  Rust

https://readrust.net/ - *https://twitter.com/read_rust*

Curated posts from the Rust programming language community

Not officially supported

Podcasts

The Request for Explanation Podcast

-

Discussion about RFCs

Rusty Spike Podcast

-

A podcast for Rust and Servo

https://newrustacean.com/

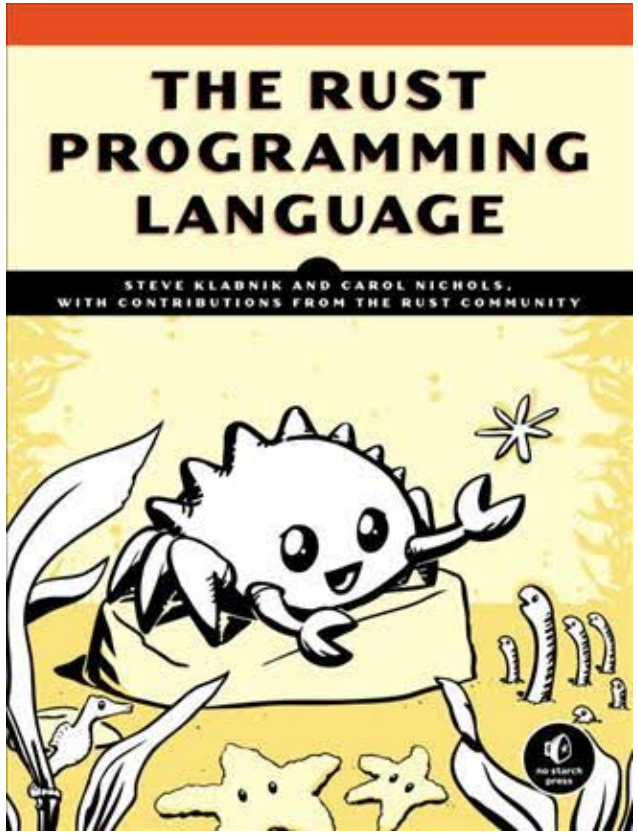
https://request-for-explanation.github.io/podcast/

https://rusty-spike.blubrry.net/

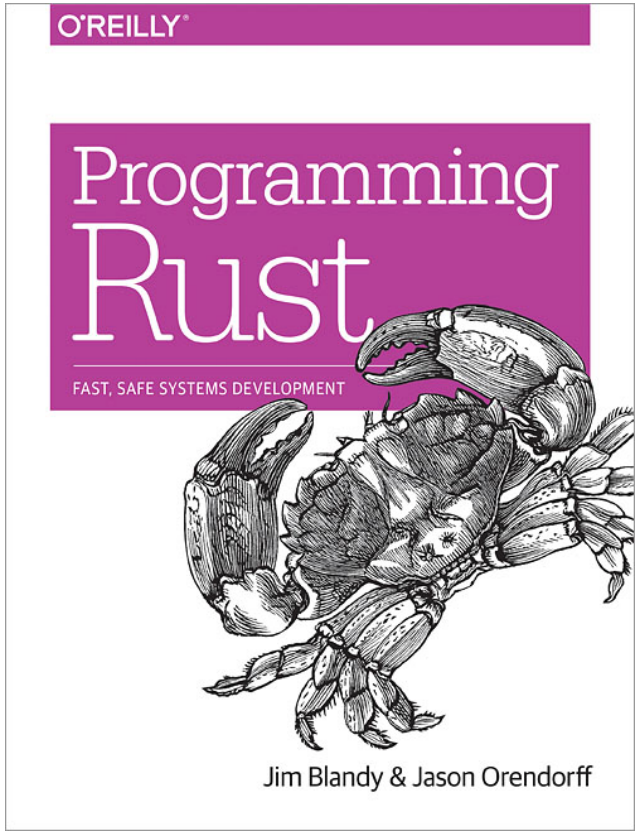
Learning resources

- https://rustbyexample.com/
- https://github.com/carols10cents/rustlings
- Rust 101: https://www.ralfj.de/projects/rust-101/main.html
- Into_rust() – screencast episodes: http://intorust.com/
- Ashley Williams brief intro to the syntax: https://ashleygwilliams.github.io/a-very-brief-intro-to-rust/#1
- GitHub repository referencing them: https://github.com/ctjhoa/rust-learning

Books



Official book
Available online in two versions
Final version not finished



Release date: December 2017


```rust
#[macro_use]
extern crate ferris_print;

fn main() {
    ferrisprint!("Merci!");
}
```

Ferris-print macro: https://github.com/kimond/ferris-print