Rust Workshop

Philipp Krones

This is me

- Philipp Krones
  - GitHub: @flip1995
  - Member of The Rust Programming Language Organization
  - Team Lead of Clippy (Rust Linter)
  - Background in Software/Compiler Engineering and AI

Get the Repo

- Fork: `embecosm/rust-workshop`
- Clone the fork:
  - SSH: `git clone git@github.com:<username>/rust-workshop`
  - HTTP: `git clone https://github.com/<username>/rust-workshop`

Contents

Ecosystem

Commands that should work

```
$ rustup -V
rustup 1.23.1 (3df2264a9 2020-11-30)
$ cargo -V
cargo 1.48.0 (65cbdd2dc 2020-10-14)

$ cargo new hello_world
$ cd hello_world
$ cargo run
"Hello, world!"
```

Works for me!

A) Hurray! :)

B) Nay! :(

Rustup

- The Rust Toolchain Manager

- Used for:
  - Installing Toolchains: stable, beta, nightly, ...
  - Installing Components: rustfmt, clippy, rust-analyzer, ...
  - Installing Targets: `riscv32imac-unknown-none-elf`, ...

Cargo

- The Rust Package Manager

- Used for:
  - Compiling & Testing your code
  - Automatically downloading dependencies
  - Installing Rust programs from the package registry

- Can be compared to NPM

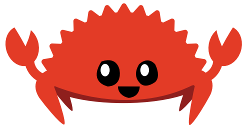
Tools

- **Rustfmt:** Code Formatting
- **Clippy:** Static Code Analysis

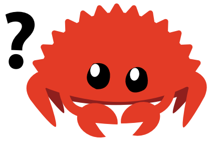
And Now You: `00-rustfmt`

Completed?

A) Yay!



B) Need help!



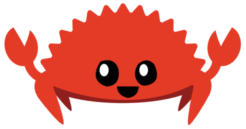
Rust Error Messages

```
error[E0308]: mismatched types
  --> src/main.rs:15:20
   |
15 |     let y: usize = 0u16;
   |            -----   ^^^^ expected `usize`, found `u16`
   |            |
   |            expected due to this
   |
help: change the type of the numeric literal from `u16` to `usize`
   |
15 |     let y: usize = 0usize;
   |                     ^^^^^^
```

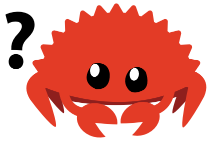
And Now You: `01-clippy`

Completed?

A) Yay!



B) Need help!



Basic Rust

Basic Rust: Syntax

```
fn foo(arg: i32, b: bool) -> u32 {
  if b {
    return arg as u32;
  } else {
    println!("arg = {}", arg);
  }
  (arg + 1) as u32
}
```

Basic Rust: Main and Unit-Type ()

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

Basic Rust: Main and Unit-Type ()

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

Basic Rust: Everything is an Expression

```rust
fn bar(b: bool) -> i32 {
  let x = if b {
    42
  } else {
    21
  };

  x
}
```

Basic Rust: Primitive Types

- `u8, u16, u32, u64, u128, usize`
- `i8, i16, i32, i64, i128, isize`
- `bool`
- `char (4 byte)`
- `&str ("slice"), String ("owned")`
- `&[T] ("slice"), Vec<T> ("owned")`

Basic Rust: Ownership

```rust
fn take(s: String) { /* Code using `s` */ }

fn main() {
  let s = String::from("Rust is cool!");
  take(s);
  //   ^--------------------------------|
  println!("{}", s); // Error: `s` was moved
}
```

Basic Rust: References / Borrowing

```rust
fn borrow(s: &String) { /* Code using `s` */ }

fn main() {
  let s = String::from("Rust is cool!");
  borrow(&s);
  //     ^----------------------|
  println!("{}", s); //         |
  // No error: only a reference `&` to `s` was
  // passed into `borrow()`.
}
```

Basic Rust: Clone

```
fn take(s: String) { /* Code using `s` */ }

fn main() {
  let s = String::from("Rust is cool!");
  take(s.clone());
  //      ^^^^^^^--------------------|
  println!("{}", s); //              |
  // No error: `s` was explicitely `clone()`d
  // before it was passed into `take()`.
}
```

Basic Rust: Copy Types

```rust
fn take(x: u32) { /* Code using `x` */ }

fn main() {
  let x = 42u32;
  take(x);
  println!("{}", x);
  // No error: `x` is `Copy`, meaning that `x`
  // gets implicitly copied before it is moved
  // into `take()`.
}
```

Basic Rust: Mutable Reference

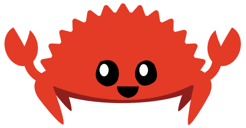
```
fn mutate(s: &mut String) { /* Mutate `s` */ }

fn main() {
  let mut s = String::from("Rust is cool!");
  //  ^^^ Everything is immutable by default.
  let ref_s = &s;
  mutate(&mut s);
  // Error: `s` cannot be borrowed mutable and
  // immutable at the same time.
}
```

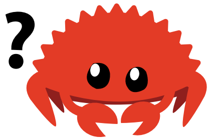
Basic Rust: Mutable References

```rust
fn mutate(s: &mut String) { /* Mutate `s` */ }

fn main() {
  let mut s = String::from("Rust is cool!");
  mutate(&mut s); // mutable borrow dropped here
  //           ^------------------------|
  let ref_s = &s;
  // No error: The mutable borrow was dropped
  // before the immutable borrow
}
```

And Now You: `02-basics`

Completed?

A) Yay! 

B) Need help! 

Interlude: IDE Support

- `rust-analyzer`: Implementation of the Language Server Protocol (LSP)
- Install:
  - VSCode: rust-analyzer extension (press "Yes" when it asks you to download the binary)
  - (Neo)Vim: `coc.nvim` (`:CocInstall coc-rust-analyzer`), but you have to install the binary yourself.
  - Emacs: `lsp-mode` (so I was told) or these instructions, but you have to install the binary yourself.

Pattern Matching

Pattern Matching: `structs`

```rust
struct S {
  a: u32,
  b: bool,
}
let s = S { a: 42, b: true };
println!("{}", s.a);

struct T(u32, bool);
let t = T(42, true);
println!("{}", t.0);
```

Pattern Matching: Methods

```rust
struct T(u32, bool);

impl T {
  fn new(a: u32, b: bool) -> Self {
    Self(a, b)
  }
  fn into_tuple(self) -> (u32, bool) {
    (self.0, self.1)
  }
}
```

Pattern Matching: **enums**

```rust
#[derive(Debug)]
enum E {
  A,
  B(u32),
  C { a: u32 },
}
let a = E::A;
let b = E::B(42);
let c = E::C { a: 42 };

println!("{a} {b} {c}", a=a, b=b, c=c);
```

Pattern Matching: `matches`

```
enum E { A, B(u32), C { a: u32 } }

match e {
  E::A => println!("A");
  E::B(_) => println!("B");
  // Error: Not all variants covered
}
```

Pattern Matching: **`matches`**

```
enum E { A, B(u32), C { a: u32 } }

match e {
  E::A => println!("A");
  E::B(_) => println!("B");
  E::C { .. } => println!("C");
}
```

Pattern Matching: `matches`

```rust
enum E { A, B(u32), C { a: u32 } }

match e {
  E::A => println!("A");
  E::B(_) => println!("B");
  _ => println!("default");
//^ Wildcard: Catch all remaining variants
}
```

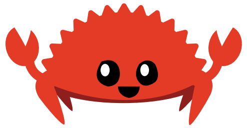
Pattern Matching: `if let`

```
enum E { A, B(u32), C { a: u32 } }

if let E::B(16) = e {
  println!("e is E::B with value 16");
}
```

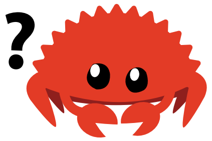
And Now You: `03-pattern-matching`

Completed?

A) Yay!



B) Need help!



Error Handling

Error Handling: `Option` and `Result`

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

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

Error Handling: Useful Methods

```
let opt = Some(42);
let res = Ok(42);

opt.unwrap();         // Take value or panic/abort
opt.unwrap_or(0);  // Take value or use default

// Convert to a different type
opt.map(|x| x == 42);
// Convert to a different type or use default
opt.map_or(false, |x| x == 42);

opt.ok_or("Error!");  // Convert to a `Result`
res.ok();                    // Convert to an `Option`
```

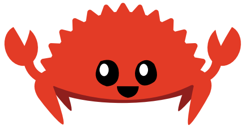
Error Handling: Propagate Error

```rust
fn foo() -> Result<usize, &str> {
  /* some code */
}

fn bar() -> Result<(), &str> {
  let res = foo()?;
  //              ^ Unwraps `Result` or
  //                propagates `Err` to caller
  if res > 10 { Ok(()) } else { Err("err") }
}
```

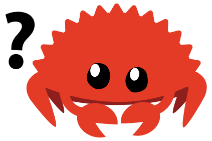
And Now You: `04-error-handling`

Completed?

A) Yay!



B) Need help!



Documentation:

- `Option<T>`
- `Result<T, E>`

Iterators

Iterators: Closures

```
fn foo<F>(f: F) -> bool
where
  F: Fn(usize) -> bool,
{
  f(42)
}
fn is_even(x: usize) -> bool { x % 2 == 0 }

foo(|x: usize| -> bool { x == 42 });
foo(|x| x == 42);
foo(is_even);  // same as: `foo(|x| is_even(x))`
```

Iterators: `for`-Loops

```rust
for elem in vec.iter() {
  println!("{}", elem);
}

for (i, elem) in vec.iter().enumerate() {
  println!("vec[{}] = {}", i, elem);
}

for elem in vec.iter().filter(|x| x % 2 == 0) {
  println!("Even: {}", elem);
}
```

Iterators: `while`-Loops

```rust
while x < 10 {
  modify_x(&mut x, y);
}

let mut it = vec.iter();

while let Some(elem) = it.next() {
  println!("{}", elem);
}
```

Iterators: Useful Methods

```
let it: Iterator<Item = T1> = vec.iter();
let it2: Iterator<Item = T2> = vec2.iter();

it.enumerate(); // converts to: `Item = (usize, T1)`
it.map(|x| x % 2 == 0); // converts to: `Item = bool`
it.filter(|x| x > 3); // only elements that are `> 3`
it.chain(it2); // new iterator: `it`→`it2`, `T1 == T2`
it.zip(it2); // new iterator: `Item = (T1, T2)`
it.any(|x| x == 0); // returns `bool`
it.all(|x| x > 0); // returns `bool`
iter::empty(); // returns empty iterator
iter::once(42); // returns iterator with one element
```

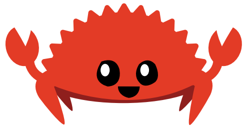
Iterators: Laziness

```rust
let it = vec.iter();

it.filter(|x| x > 3);
// Warning: "iterators are lazy and do nothing
//           unless consumed"

let greater_three = it.filter(|x| x > 3)
                      .collect::<Vec<_>>();
// Turbofish `::<>`           ^^^^^^^^^^
```

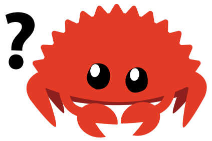
And Now You: `05-iterators`

Completed?

A) Yay!



B) Need help!



Documentation:

- `Iterator`

Good Practices

Good Pratices: Documentation

```rust
//! Module Documentation

#![warn(missing_docs)] // warn if undocumented

/// Item Documentation
pub fn foo() {}
```

Good Practices: Documentation Structure

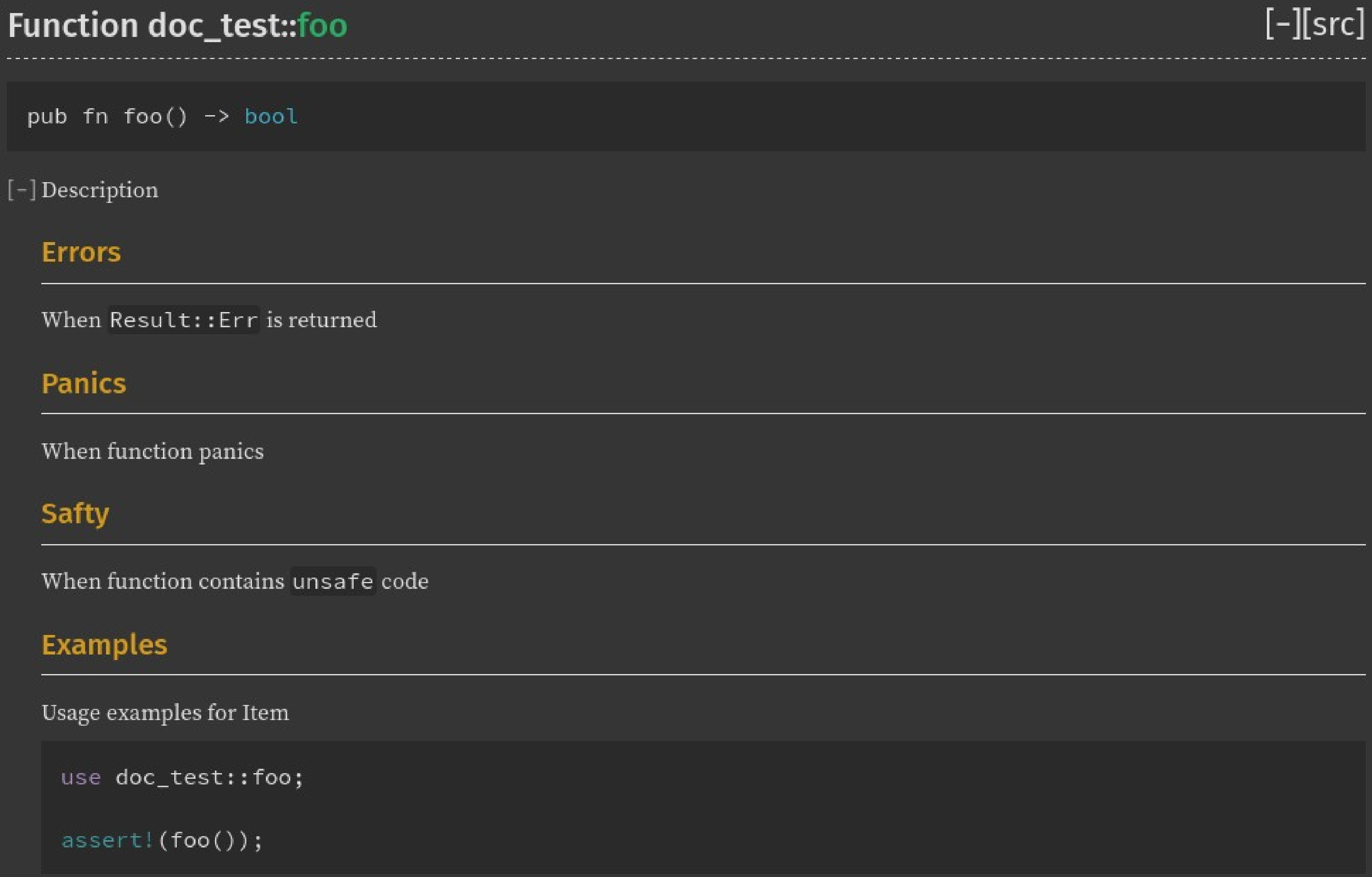
```rust
/// Description
/// # Errors
/// When `Result::Err` is returned
/// # Panics
/// When function panics
/// # Safty
/// When function contains `unsafe` code
/// # Examples
/// Usage examples for Item
pub fn foo() {}
```

Good Practices: doc-Tests

```rust
/// # Examples
///
/// ```rust
/// use doc_test::foo;
///
/// assert!(foo());
/// ```
pub fn foo() -> bool { true };
```

Good Practices: **doc-Tests**

```
   Doc-tests doc_test

running 1 test
test src/lib.rs - foo (line 19) ... ok

test result: ok. 1 passed; 0 failed; 0 ignored;
0 measured; 0 filtered out
```



Function doc_test::foo

[−][src]

```
pub fn foo() -> bool
```

[−] Description

Errors

When `Result::Err` is returned

Panics

When function panics

Safty

When function contains `unsafe` code

Examples

Usage examples for Item

```
use doc_test::foo;

assert!(foo());
```

Good Practices: Modules

```
$crate
├── Cargo.lock
├── Cargo.toml
├── README.md
└── src
    ├── lib.rs          // `$crate` (library root)
    ├── main.rs         // binary root
    ├── nested          // nested module
    │   ├── mod.rs      // `$crate::nested` (module root)
    │   ├── one.rs      // `$crate::nested::one`
    │   └── two.rs      // `$crate::nested::two`
    └── some_module.rs  // `$crate::some_module`
```

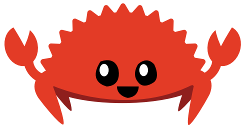
Good Practices: Modules (`lib.rs`)

```rust
// will be available outside the crate (public)
pub mod some_module;
// will only be availabe in the crate (private)
mod nested;
// Item `foo` will be available outside the
// crate with the path `$crate::foo` (re-export)
pub use nested::foo;
// The use statement aboth allows the usage of
// `foo` without a fully qualified path
fn run() { foo() }  // same as `nested::foo()`
```

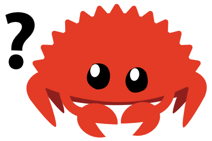
And Now You: `06-good-practices`

Completed?

A) Yay!



B) Need help!



Generics

Generics: `structs`

```rust
struct S<T> {
  v: Vec<T>,
}

impl<T> S<T> {
  fn new(elem: T) -> Self {
    Self { v: vec![elem] }
  }
}
```

```rust
enum E<T> {
  A(T),
}
```

Generics: `fns`

```rust
fn identity<T>(x: T) -> T {
  x
}
```

Generics: Bounds

```
fn foo<T: Display>(x: T) {
  println!("{}", x);
}

fn foo<T>(x: T)
where
  T: Display,
{
  println!("{}", x);
}
```

Generics: Lifetimes

```rust
struct S<'a> {
  s: &'a str,
}

impl<'a> S<'a> {
  fn new(s: &'a str) -> Self {
    Self { s }
  }
}
```

Generics: Lifetimes

```
struct S<'a> {
  s: &'a str,
}

impl S<'_> {
  fn print_me(&self) -> Self {
    println!("{}", self.s);
  }
}
```

Generics: Lifetimes

```rust
struct S<'a> { s: &'a str }

let mut s = S { s: "my str" };
{
  let new_s = String::from("my new str");
  s.s = &new_s;
  //    ^^^^^^ Error: Does not live long enough
} // `new_s` dropped here: end of scope
println!("{}", s.s);
//             ^^^ borrow later used here
```

Generics: Lifetimes

```
error[E0597]: `new_s` does not live long enough
 --> src/main.rs:6:44
  |
6 |         s.s = &new_s;
  |               ^^^^^^ borrowed value does not live long enough
8 |     }
  |     - `new_s` dropped here while still borrowed
9 |     println!("{}", s.s);
  |                    --- borrow later used here

error: aborting due to previous error

For more information about this error, try `rustc --explain E0597`.
```

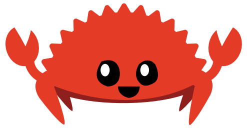
Generics: Lifetimes

```rust
struct S<'a> { s: &'a str }

let mut s = S { s: "my str" };
{
  let new_s = String::from("my new str");
  s.s = &new_s;
  println!("{}", s.s);
}
```

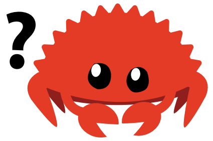
And Now You: `07-generics`

Completed?

A) Yay!



B) Need help!



Traits

Traits: Definition and Implementation

```rust
struct S;
trait T {
  fn foo(&self);
  fn bar(&self) { self.foo(); }
}

impl T for S {
  fn foo(&self) { /* some code */ }
  // optional:
  fn bar(&self) { self.foo(); self.foo(); }
}
```

Traits: `std`-Traits

```rust
struct S(u32);

impl std::ops::Add for S {
  type Output = u32;

  fn add(self, other: Self) -> Self::Output {
    self.0 + other.0
  }
}

assert_eq!(S(10) + S(1), 11u32);
```

Traits: `std`-Traits

```rust
struct S(u32);

impl std::iter::Iterator for S {
  type Item = u32;

  fn next(&mut self) -> Option<Self::Item> {
    Some(self.0)
  }
}

assert_eq!(S(10).next(), Some(10u32));
assert!(S(0).any(|x| x == 0));
```

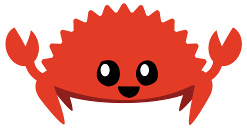
Traits: `derive`

```rust
#[derive(Debug, Eq, PartialEq)]
struct S(u32);

let s1 = S(10);
let s2 = S(10);

// By deriving `Eq, PartialEq` the `==` operator can be
// used
assert!(s1 == s2);
// By deriving `Debug`, the struct can be debug-printed
println!("{:?}", s1);
```

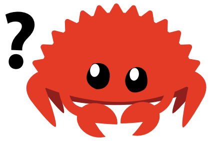
And Now You: `08-traits`

Completed?

A) Yay!



B) Need help!

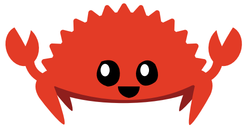


First Project

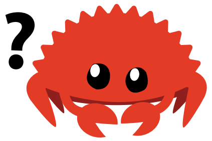
And Now You: `09-first-project`

Completed?

A) Yay!



B) Need help!



Documentation:

- `std`
- `HashMap`
- `Entry`

Part 2

Contents

Advanced Rust

`10-advanced-rust`

Advanced Rust: Contents

- Macros
- Trait Objects
- `impl Trait`
- Lifetime Bounds
- Higher-Ranked Trait Bounds
- unsafe Code

Macros

Advanced Rust: Macros

```
macro_rules! my_macro {
  ($e:expr, [$($v:ident),*]) => {
    $e;
    let v = vec![$(stringify!($v)),*];
    println!("{:?}", v);
  }
}

my_macro!(2 + 3, [a, b, c]);
```

Advanced Rust: Macro Expansion

```
macro_rules! my_macro {
  ($e:expr, [$($v:ident),*]) => {
    $e;
    let v = vec![$(stringify!($v)),*];
    println!("{:?}", v);
  }
}
my_macro!(2 + 3, [a, b, c]);
$ rustc +nightly -Zunpretty=expanded main.rs
2 + 3;
let v = vec![
  stringify!(a),
  stringify!(b),
  stringify!(c),
];
println!("{:?}", v);
```

EMBECOSM®

Advanced Rust: Macro Expansion

```rust
2 + 3;
let v = <[_]>::into_vec(box ["a", "b", "c"]);
{
  ::std::io::_print(::core::fmt::Arguments::new_v1(
    &["", "\n"],
    &match (&v,) {
      (arg0,) => [
        ::core::fmt::ArgumentV1::new(
          arg0,
          ::core::fmt::Debug::fmt
        )
      ],
    },
  ));
};
```

Trait Objects

Advanced Rust: Trait Objects

```rust
#[derive(Default)] struct VecWrap(Vec<Box<dyn T>>);
struct S1; struct S2;
trait T { fn foo(&self) -> u32; }
impl T for S1 { fn foo(&self) -> u32 { 1 } }
impl T for S2 { fn foo(&self) -> u32 { 2 } }
fn bar(v: Vec<Box<dyn T>>) -> u32 {
    v.iter().fold(0, |acc, x| acc + x.foo())
}

let mut v = VecWrap::default();
v.push(Box::new(S1));
v.push(Box::new(S2));
assert_eq!(bar(v.0), 3);
```

Advanced Rust: Trait Objects

- Pro:
  - Code Size
  - Different Types in one Collection
- Con:
  - Dynamic Dispatch → Slower during Runtime
  - Only for "Object Safe" traits
    - Return type isn't `Self`
    - There are no generic type parameters

`impl Trait`

Advanced Rust: `impl Trait`

```rust
// reminder solution chapter 8
fn animal_things<A, B>(a1: &mut A, a2: &mut B)
where
  A: AnimalBehavior,
  B: AnimalBehavior,
{
  animal_1.annoy(animal_2);
}

let mut cat = …; let mut dog = …;
animal_things(&mut cat, &mut dog);
```

Advanced Rust: `impl Trait`

```
fn animal_things(
  a1: &mut impl AnimalBehavior,
  a2: &mut impl AnimalBehavior,
) {
  animal_1.annoy(animal_2);
}

let mut cat = …; let mut dog = …;
animal_things(&mut cat, &mut dog);
```

Advanced Rust: `impl Trait` - Common usage

```
fn evens(v: &[u32])
        -> impl Iterator<Item = &u32> {
  v.iter().filter(|&x| x % 2 == 0)
}

let v = vec![1, 2, 3];
assert_eq!(evens(&v).collect::<Vec<_>>(), vec![&2]);
```

Advanced Rust: `impl Trait`

- Pro:
  - Static Dispatch
  - Syntactic sugar for Trait Bounds
- Con:
  - Code Size due to monomorphization

Lifetime Bounds

Advanced Rust: Lifetime Bounds

```rust
fn foo<'a, 'b: 'a>(s1: &'a str) -> &'b str {
  s1
}
```

- Error: "lifetime of reference outlives lifetime of borrowed content"
- `'b: 'a` means: `'b` outlives `'a`
- But `s1` has lifetime `'a`, which doesn't necessarily live longer than `'b`

```rust
fn foo<'a, 'b: 'a>(s1: &'b str) -> &'a str {
  s1
}
```

Higher-Ranked Trait Bounds

EMBECOSM®

Advanced Rust: Higher-Ranked Trait Bounds

```rust
fn foo<F>(f: F) -> i32
where
  F: Fn(&i32, &i32) -> &i32
{
  let (x, y) = (0, 1);
  *f(&x, &y)
}

assert_eq!(foo(|x, _| x), 0);
```

Advanced Rust: Higher-Ranked Trait Bounds

```
error[E0106]: missing lifetime specifier
 --> src/main.rs:6:44
  |
4 |   F: Fn(&i32, &i32) -> &i32
  |         ----  ----     ^ expected named lifetime parameter
  |
  = help: this function's return type contains a borrowed value, but
the signature does not say whether it is borrowed from argument 1 or
argument 2
  | ...

error: aborting due to previous error

For more information about this error, try `rustc --explain E0106`.
```

Advanced Rust: Higher-Ranked Trait Bounds

```rust
fn foo<F>(f: F) -> i32
where
  F: for<'a> Fn(&'a i32, &i32) -> &'a i32
{
  let (x, y) = (0, 1);
  *f(&x, &y)
}

assert_eq!(foo(|x, _| x), 0);
```

`unsafe` Code

Advanced Rust: `unsafe` Code

- Dereference a raw pointer
- Call an `unsafe` function or method (FFI)
- Access or modify a mutable static variable
- Implement an `unsafe` trait
- Access fields of `unions`

Unsafe code does not disable the borrow checker!

Advanced Rust: `unsafe` Code – Pointer Deref

```
unsafe fn deref(x: *const u32) -> u32 {
  *x
}
```

Advanced Rust: unsafe Code – Function Call

```rust
unsafe fn deref(x: *const u32) -> u32 {
  *x
}

let x = 42u32;
// Creating a raw pointer is safe!
let ptr_x = &x as *const u32;
//             ^ don't forget this!
assert_eq!(unsafe { deref(ptr_x) }, 42);
```

Advanced Rust: `unsafe` Code – `static mut`

```rust
static mut COUNTER: u32 = 0;

fn add_to_count(inc: u32) {
    unsafe { COUNTER += inc; }
}

add_to_count(3);
assert_eq!(unsafe { COUNTER }, 3);
```

Advanced Rust: `unsafe` Code – `unsafe` Trait

```rust
struct S(Rc<u32>);

// Guarantee that S can safely be sent across
// threads. Think before doing this!
unsafe impl Send for S {}

// Guarantee that S can safely be accessed across
// threads. Think before doing this!
unsafe impl Sync for S {}
```

Advanced Rust: `unsafe` Code – Access `unions`

```rust
union MyUnion {
  f1: u32,
  f2: f32,
}
// Creating a `union` is safe
let u = MyUnion { f1: 1 };
// Accessing its fields not
let f = unsafe { u.f2 };

println!("{}", f);
```

Embedded Rust

Embedded Rust: Platform Support

- Platform Support
- RISC-V Support

Target	std	host	Notes
riscv32i-unknown-none-elf	*		Bare RISC-V (RV32I ISA)
riscv32imac-unknown-none-elf	*****		**Bare RISC-V (RV32IMAC ISA)**
riscv32imc-unknown-none-elf	*		Bare RISC-V (RV32IMC ISA)
riscv64gc-unknown-linux-gnu	✓	✓	RISC-V Linux (kernel 4.20, glibc 2.29)
riscv64gc-unknown-none-elf	*		Bare RISC-V (RV64IMAFDC ISA)
riscv64imac-unknown-none-elf	*		Bare RISC-V (RV64IMAC ISA)

- host: A ✓ indicates that rustc and cargo can run on the host platform.
- std:
  - ✓ indicates the full standard library is available.
  - * indicates the target only supports `no_std` development.

Embedded Rust: `no_std`

Feature	`no_std`	`std`
heap (dynamic memory)	*	✓
collections (`Vec`, `HashMap`, etc.)	**	✓
stack overflow protection	✘	✓
runs init code before main	✘	✓
libstd available	✘	✓
libcore available	✓	✓
writing firmware, kernel, or bootloader code	✓	✘

* Only if you use the `alloc` crate and use a suitable allocator.
** Only if you use the `collections` crate and configure a global default allocator.

Embedded Rust: `cargo generate`

- Some platforms have a template providing
  - preset linker flags
  - linker scripts
  - …
- These templates are provided by the Embedded Working Group
- Generate a project from the RISCV template:
  - `cargo install cargo-generate`
  - `cargo generate --git https://github.com/riscv-rust/riscv-rust-quickstart`

Embedded Rust: Toolchain

- `cargo-binutils` for targets that have a complete LLVM toolchain
- RISC-V: GNU binutils:
  - RISC-V toolchain for SiFive boards
- This contains a debugger, linker, ...

Embedded Rust: Program Board

- OpenOCD
- **JLink by Segger**
  - ArchLinux:
    - AUR: `jlink-software-and-documentation`
  - Ubuntu/Debian, Fedora:
    - Segger website: https://www.segger.com/downloads/jlink/
- Start the programmer
  - Jlink:

    -
      ```
      JLinkGDBServer
      -device FE310 -if JTAG -speed 4000 -port 3333 -nogui
      ```
  - OpenOCD:
    - `openocd -f sifive-hifive1.cfg`
    - `sifive-hifive1.cfg`

Embedded Rust: Compiling

```
cargo build
```

Embedded Rust: Compiling

```
$ cat .cargo/config
[target.riscv32imac-unknown-none-elf]
runner = "riscv64-unknown-elf-gdb -q -x gdb_init"
rustflags = [
  "-C", "link-arg=-Thifive1-link.x",
]

[build]
target = "riscv32imac-unknown-none-elf"
```

Embedded Rust: Blinking LED

```
cargo run --example=leds_blink
```

That's it!

Some more Stuff

More Rust

- Rust Book
- Rustonomicon
- Community
  - Welcoming, open, and diverse. Come and join!
  - Discord
    - Official
    - Community discord
  - Zulip
  - GitHub
- Rust Embedded Book
- Awesome Rust Mentors

Open Source Rust

- Compiler:
  - `rust-lang/rust`
  - `rust-lang/rust-clippy`
- IDE: `rust-analyzer/rust-analyzer`
- Web:
  - `actix/actix-web`
  - `yewstack/yew`
- Async: `tokio-rs/tokio`
- Database: `diesel-rs/diesel`
- ...and many, many more: `Crates.io`

This is me

- Philipp Krones
- E-Mail: philipp.krones@embecosm.com
- GitHub: @flip1995
- Discord: @flip1995#2683
- Zulip: @flip1995
- LinkedIn: philkrones (**not** looking for a Job)
- Website: flip1995.com (currently empty)

EMBECOSM®

Thank You!

www.embecosm.com