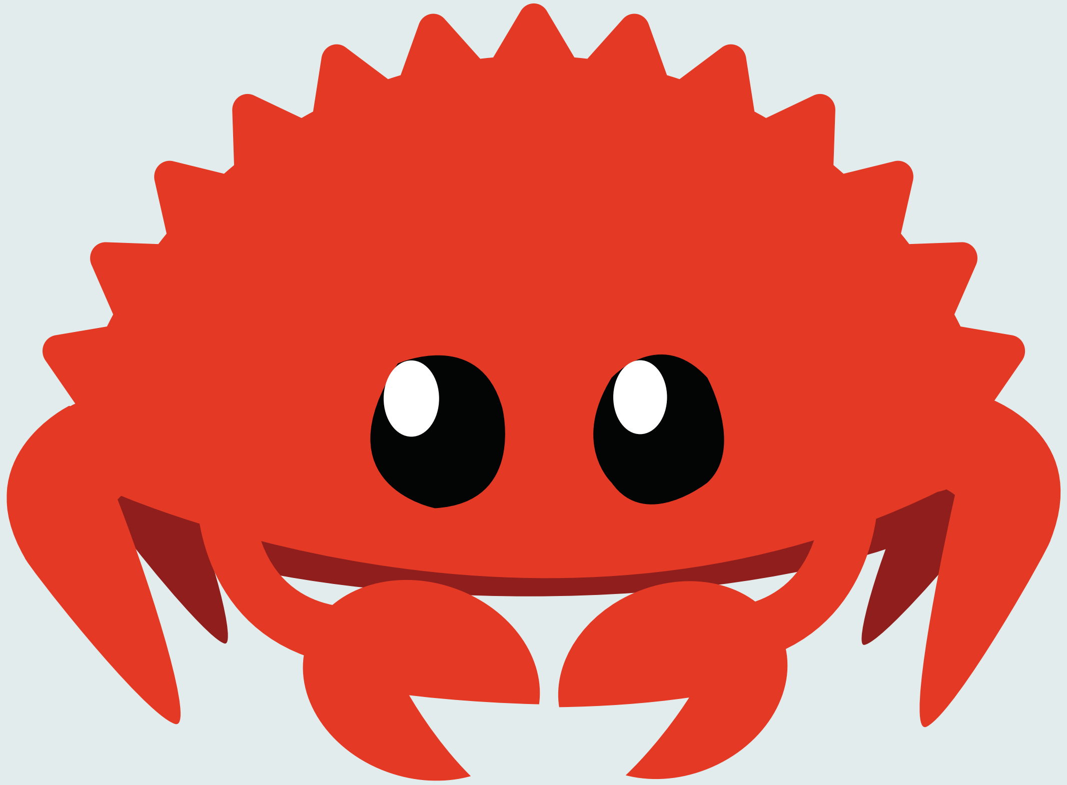
A sneak peek into* Rust

Happy times!



Martijn Gribnau

github.com/foresterre

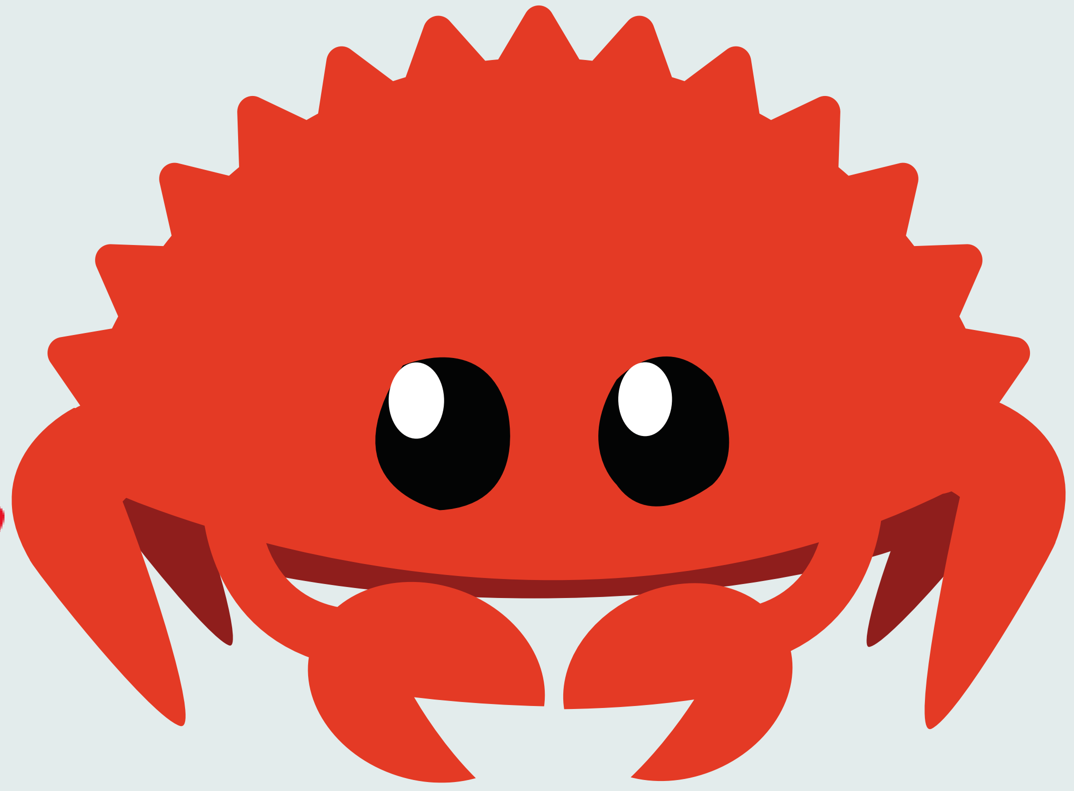
2022 // CC BY 4.0

* https://doc.rust-lang.org/std/convert/trait.Into.html 🤪

A sneak peek into Rust

Happy times!

Ferris



Ferris, the unofficial Rust mascot, designed by Karen Rustad Tölva

So, what is Rust ... (to me) ?

Rust isn't

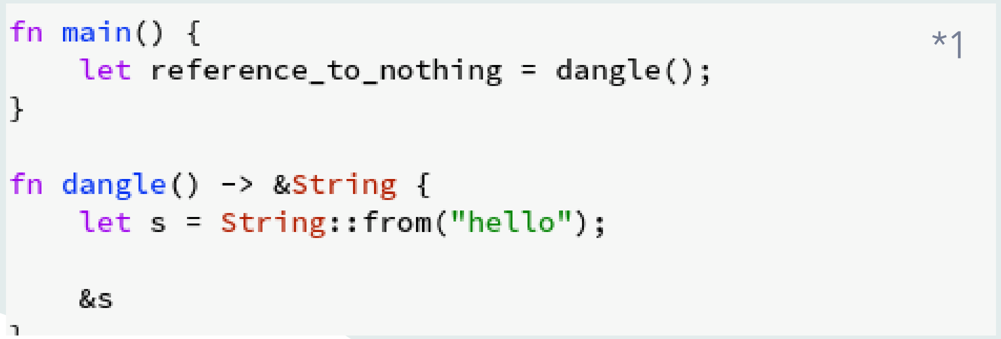
- Rust is not another C or C++
  - But borrows a lot of syntax of C/C++/Java
  - Largely targets the same market
- Rust is not Go
  - Both relatively new, but very different
- No null
- ...
- But, builds on top of many concepts from other languages

A different beast, ... yet familiar

"if I had asked people what they wanted, they would have said: faster horses"
- Henry Ford (Tris from Lost Terminal Podcast describing Rust)

Rust is: The standard introduction

- Blazingly fast systems programming language
- Memory safety
  - No dangling pointers
  - RAII like destructors
- Safe, and relatively easy parallelization
  - No data races
- Ownership & the borrow checker



*1

```
fn main() {
    let reference_to_nothing = dangle();
}

fn dangle() -> &String {
    let s = String::from("hello");

    &s
```

*1 https://doc.rust-lang.org/stable/book/ch04-02-references-and-borrowing.html?highlight=dangling#dangling-references

A quick language tour

- Time to open my IDE

Superpowered enums

- Not just a number or string
- Tagged unions, sum types

```
#[derive(Debug)]
enum Type {
    Num,
    Bool,
    List(Box<Type>),
    Func(Box<Type>, Box<Type>),
}
```

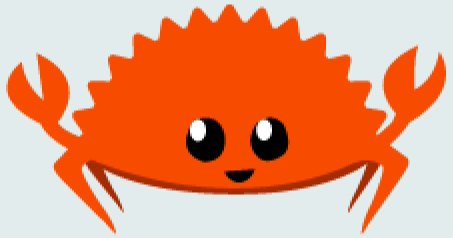
Rust is: a language of expression(s)

TypeScript

```
let payments;

if (condition) {
    payments = await byInternalIds();
} else {
    payments = await byPaymentServiceProviderIds();
}
```

Rust

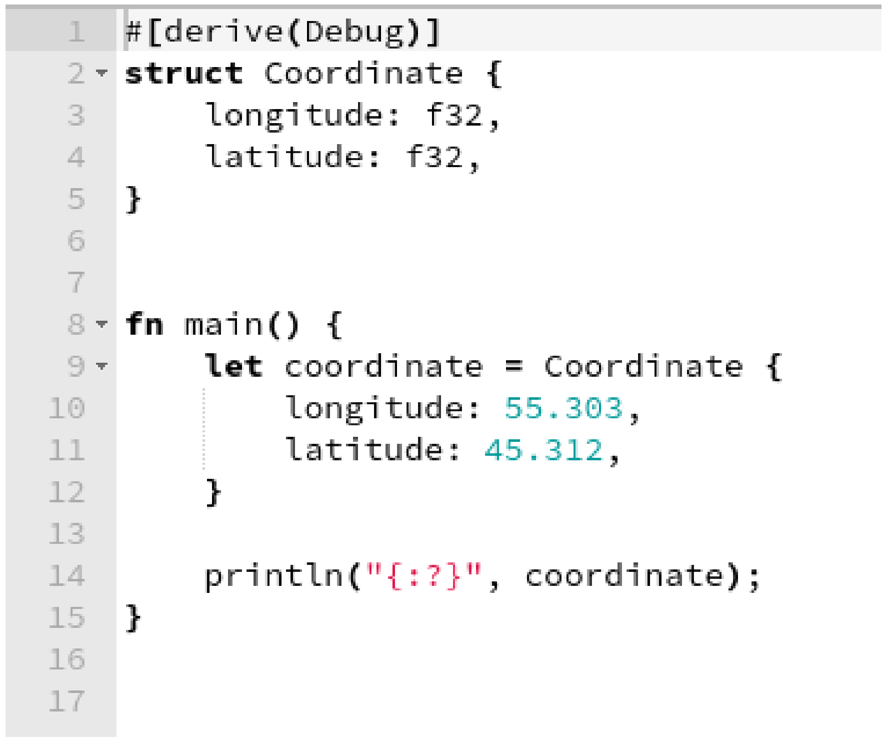
```
let payments = if condition {
    byInternalIds() // <-- Note, no ';' here!
} else {
    byPaymentServiceProviderIds() // Nor here! It's an expression!
};
```



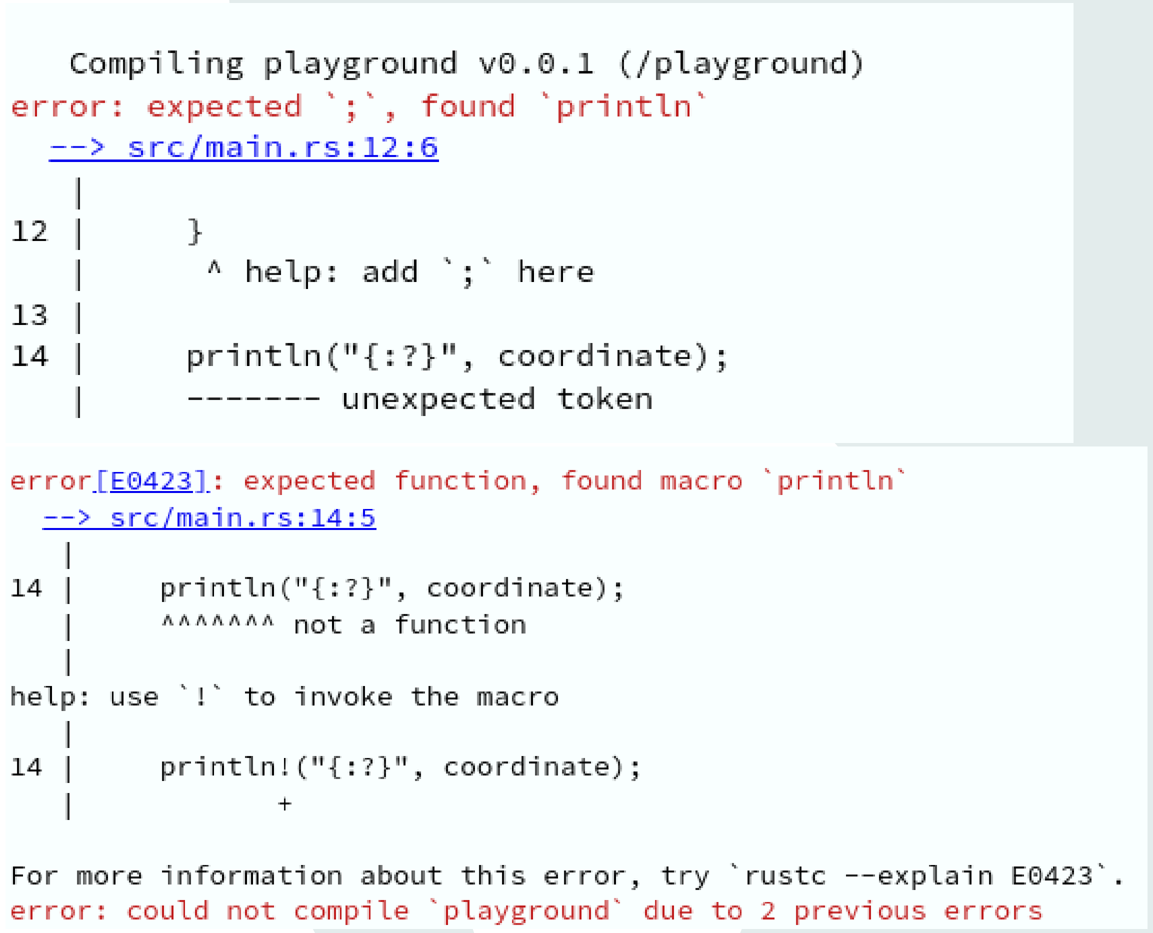
a language where

Rust is: the compiler is your friend

* Not just errors, but helpful error messages.



```
#[derive(Debug)]
struct Coordinate {
    longitude: f32,
    latitude: f32,
}


fn main() {
    let coordinate = Coordinate {
        longitude: 55.303,
        latitude: 45.312,
    }

    println("{:?}", coordinate);
}
```



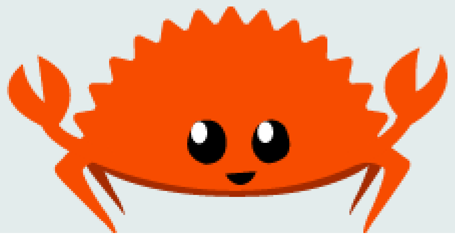
```
   Compiling playground v0.0.1 (/playground)
error: expected `;`, found `println`
  --> src/main.rs:12:6
   |
12 |     }
   |      ^ help: add `;` here
13 |
14 |     println("{:?}", coordinate);
   |     ------- unexpected token

error[E0423]: expected function, found macro `println`
  --> src/main.rs:14:5
   |
14 |     println("{:?}", coordinate);
   |     ^^^^^^^ not a function
   |
help: use `!` to invoke the macro
   |
14 |     println!("{:?}", coordinate);
   |            +

For more information about this error, try `rustc --explain E0423`.
error: could not compile `playground` due to 2 previous errors
```

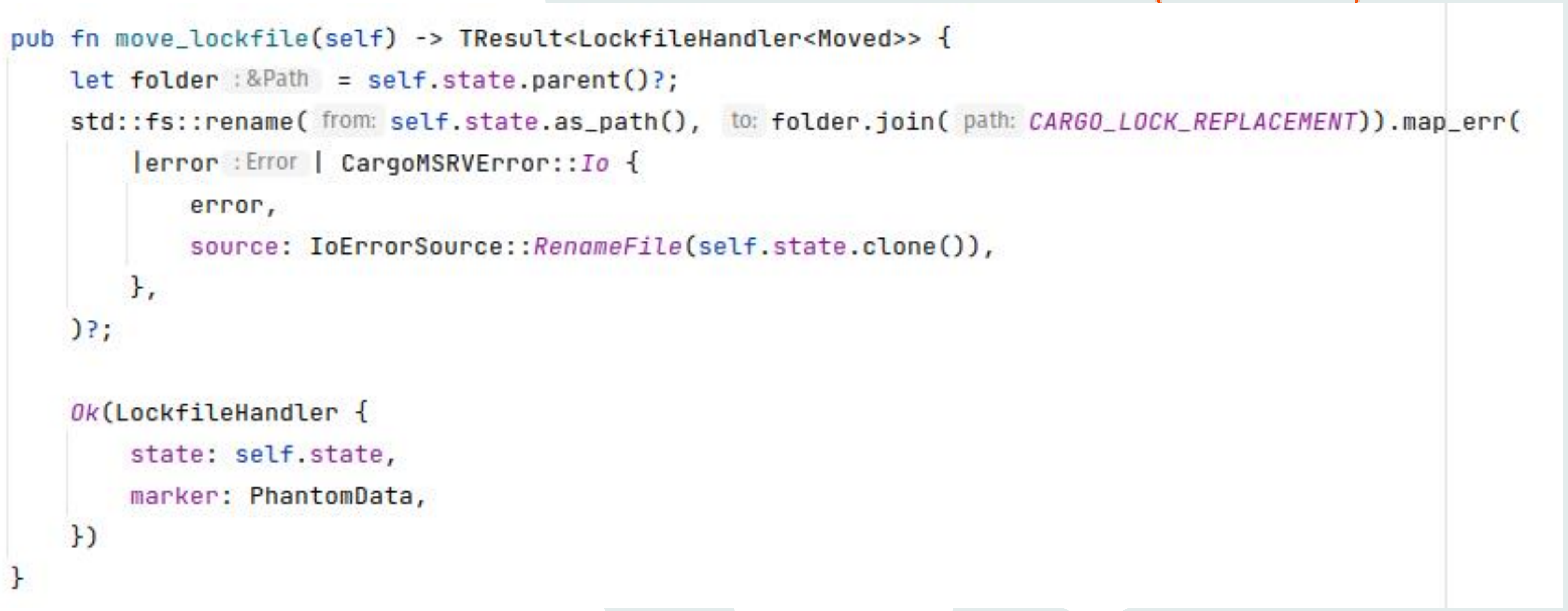
Rust is: a language where types have power

- Result<T,E> and Option<T>, or make your own :)
  - An example: error handling
- Generics
  - Monomorphization
- Traits

Error handling with Result<T, E> (1/2)

- Errors are 'just' types
- Error handling is not an after thought





```
pub fn move_lockfile(self) -> TResult<LockfileHandler<Moved>> {
    let folder : &Path = self.state.parent()?;
    std::fs::rename( from: self.state.as_path(),  to: folder.join( path: CARGO_LOCK_REPLACEMENT)).map_err(
        |error : Error | CargoMSRVError::Io {
            error,
            source: IoErrorSource::RenameFile(self.state.clone()),
        },
    )?;

    Ok(LockfileHandler {
        state: self.state,
        marker: PhantomData,
    })
}
```

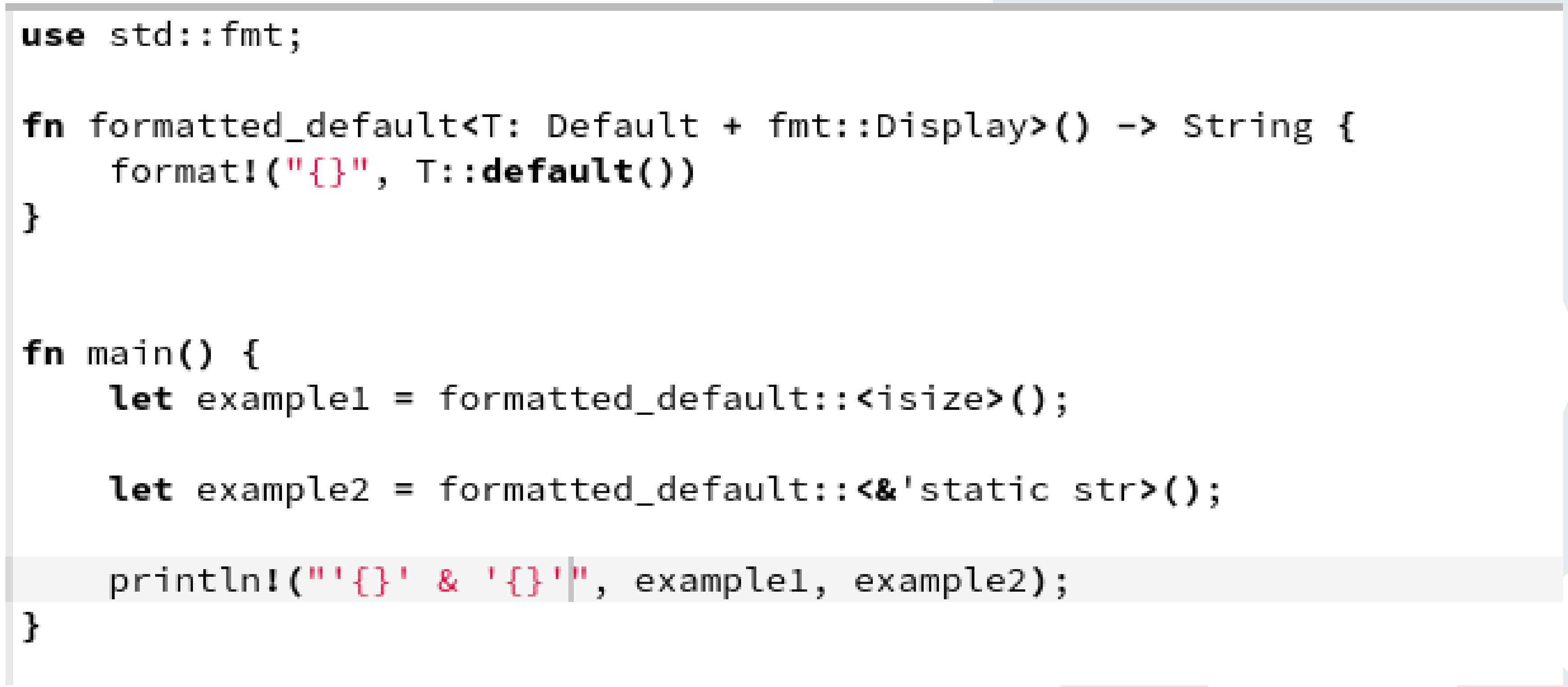
Error handling with Result<T, E> (2/2)

- Errors are 'just' types
- Error handling is not an after thought

```rust
#[derive(Debug, thiserror::Error)]
#[error("No Rust releases to check {} {} (search space: [{}])",
    min.as_ref().map(|s| format!("(min: {})", s)).unwrap_or_default(),
    max.as_ref().map(|s| format!("(max: {})", s)).unwrap_or_default(),
    search_space.iter().map(|r| r.version().to_string()).collect::<Vec<_>>().join(", ") )
]
pub struct NoToolchainsToTryError {
    pub(crate) min: Option<BareVersion>,
    pub(crate) max: Option<BareVersion>,
    pub(crate) search_space: Vec<Release>,
}
```

Generics

- Generics
  - Monomorphization



```rust
use std::fmt;

fn formatted_default<T: Default + fmt::Display>() -> String {
    format!("{}", T::default())
}


fn main() {
    let example1 = formatted_default::<isize>();

    let example2 = formatted_default::<&'static str>();

    println!("'{}' & '{}'", example1, example2);
}
```

Traits (1/2)

- Shared behaviour

```rust
/// Internal payment client
struct PaymentClient(Arc<HttpClient>);

/// PaymentServiceProvider client
struct PSPClient(Arc<HttpClient>);
```

1

```rust
trait GetPayment {
    fn get_payment(&self, id: u128) -> Result<Payment, Error>;
}
```

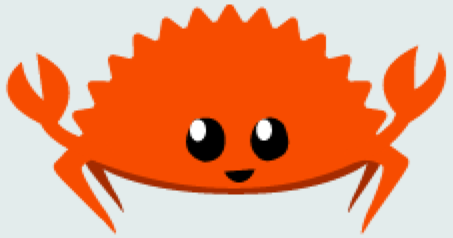
2

```rust
impl GetPayment for PaymentClient {
    fn get_payment(&self, id: u128) -> Result<Payment, Error> {
        todo!("Make it so")
    }
}

impl GetPayment for PSPClient {
    fn get_payment(&self, id: u128) -> Result<Payment, Error> {
        todo!("Make it so")
    }
}
```

3

```rust
let payment = paymentClient.get_payment(1000);
let pspPayment = pspClient.get_payment(2000);

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

4



Traits (2/2)

- Shared behaviour
- Super charged interfaces

```rust
trait PaymentStatus {
    fn status(&self, id: u128) -> Result<Status, Error>;
}
```

1

```rust
impl<T: GetPayment> PaymentStatus for T {
    fn status(&self, id: u128) -> Result<Status, Error> {
        let payment = self.get_payment(id)?;

        Ok(payment.status)
    }
}
```

2

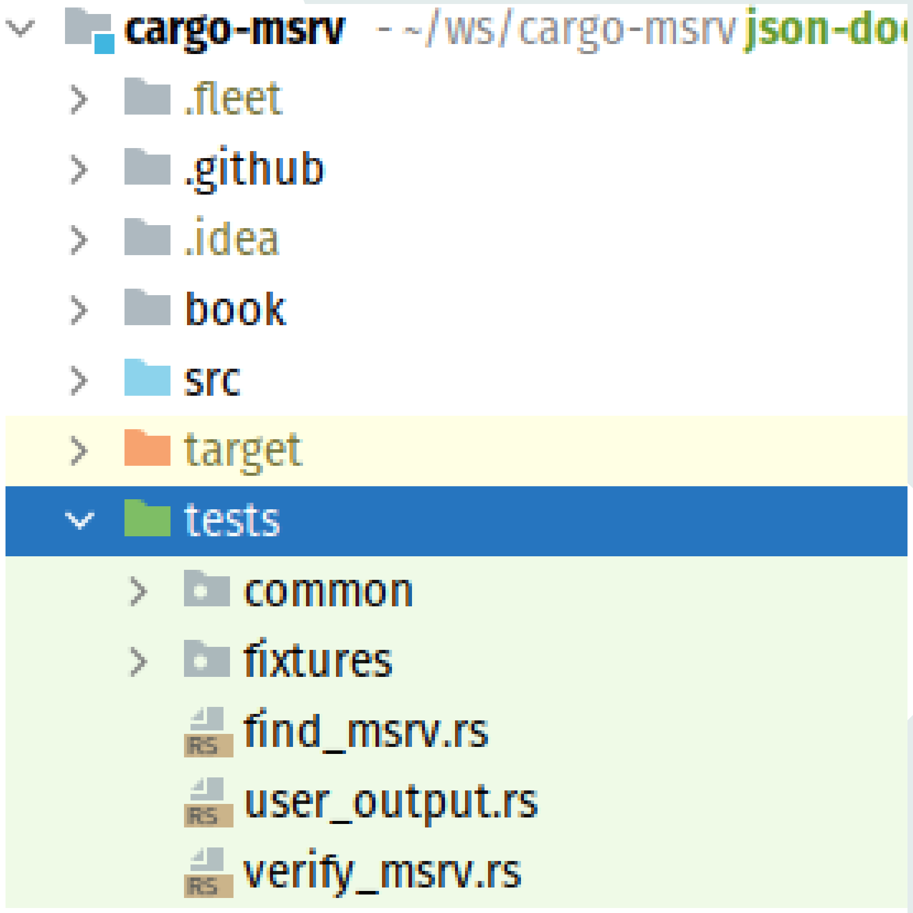
Rust is: testing first (1/3)

- Inline unit tests #[test]

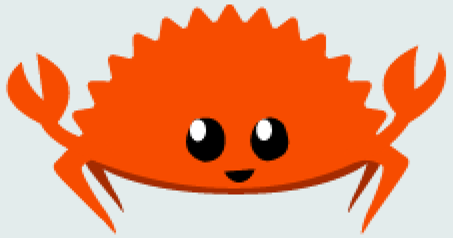
```rust
use std::fmt;

fn formatted_default<T: Default + fmt::Display>() -> String {
    format!("{}", T::default())
}


#[test]
fn zero() {
    assert_eq!(formatted_default::<isize>(), "0");
}

#[test]
fn empty_str() {
    assert_eq!(formatted_default::<&'static str>(), "");
}
```

Rust is: testing first (2/3)

- Inline unit tests #[test]
- Integration tests

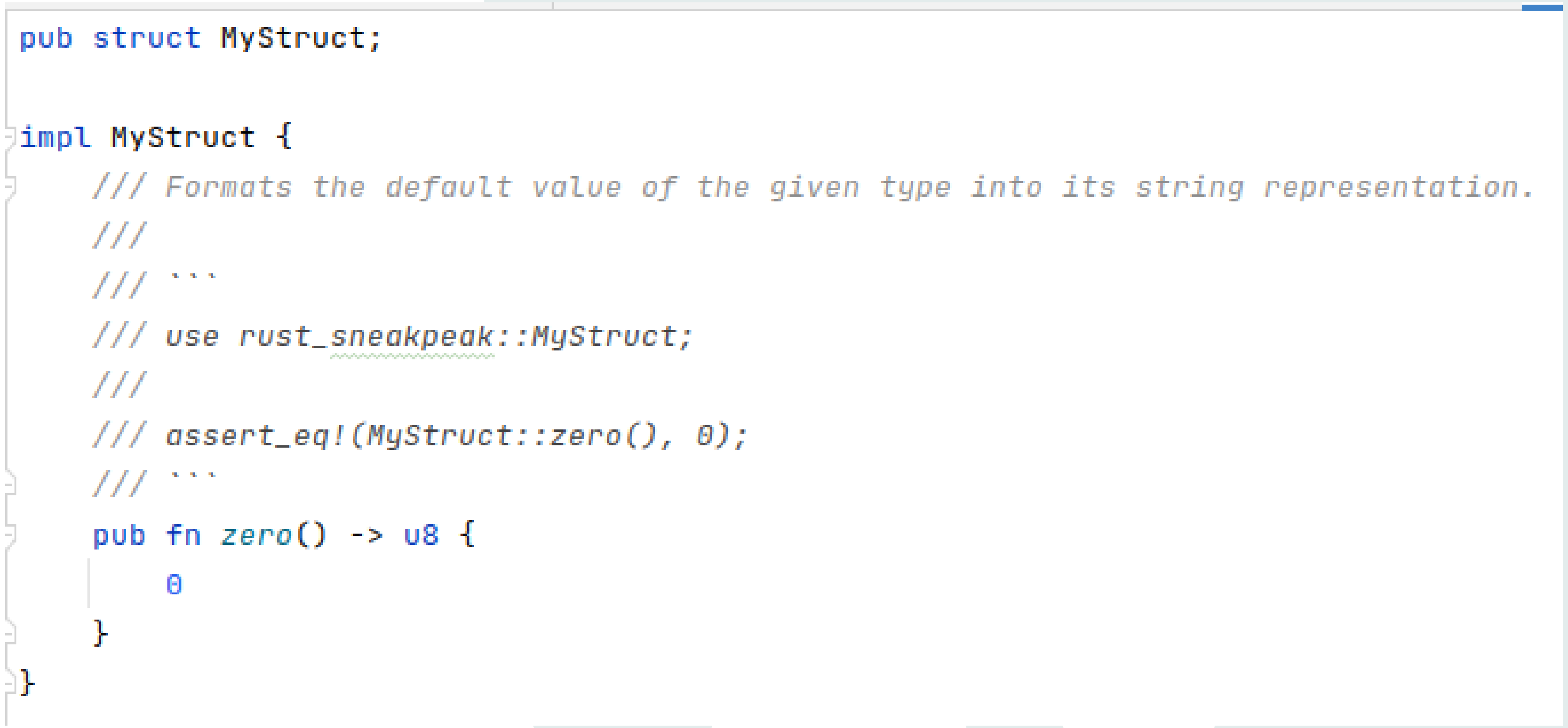


```
cargo-msrv  - ~/ws/cargo-msrv json-do
  .fleet
  .github
  .idea
  book
  src
  target
  tests
    common
    fixtures
    find_msrv.rs
    user_output.rs
    verify_msrv.rs
```



Rust is: testing first (3/3)

- Inline unit tests #[test]
- Integration tests
- Doc tests



```rust
pub struct MyStruct;

impl MyStruct {
    /// Formats the default value of the given type into its string representation.
    ///
    /// ```
    /// use rust_sneakpeak::MyStruct;
    ///
    /// assert_eq!(MyStruct::zero(), 0);
    /// ```
    pub fn zero() -> u8 {
        0
    }
}
```

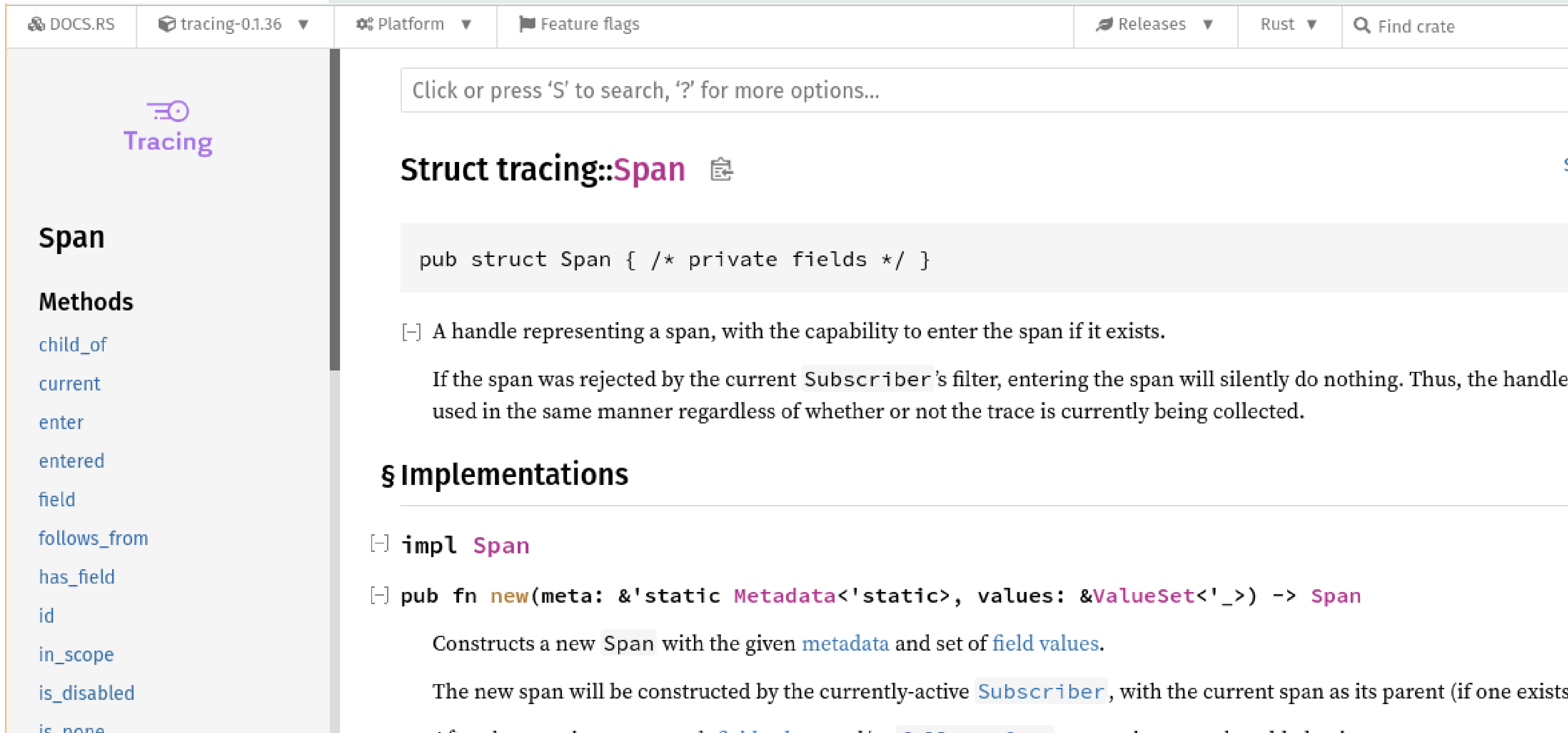
Rust is: modern tooling

- Cargo: Package & build tool
- Crates.io: packaging ecosystem
- Clippy: linter

- Rust-analyzer
- Intellij Rust

Rust is: documentation included

- Docs.rs: publicly hosted by the Rust foundation
- All crates.io packages have at least type documentation
- But usually, more
- Proper fuzzy search
  - Search by type signature



Span

Methods

child_of
current
enter
entered
field
follows_from
has_field
id
in_scope
is_disabled
is_none

Struct tracing::Span

```
pub struct Span { /* private fields */ }
```

A handle representing a span, with the capability to enter the span if it exists.

If the span was rejected by the current `Subscriber`'s filter, entering the span will silently do nothing. Thus, the handle
used in the same manner regardless of whether or not the trace is currently being collected.

§ Implementations

```
impl Span
pub fn new(meta: &'static Metadata<'static>, values: &ValueSet<'_>) -> Span
```

Constructs a new `Span` with the given metadata and set of field values.

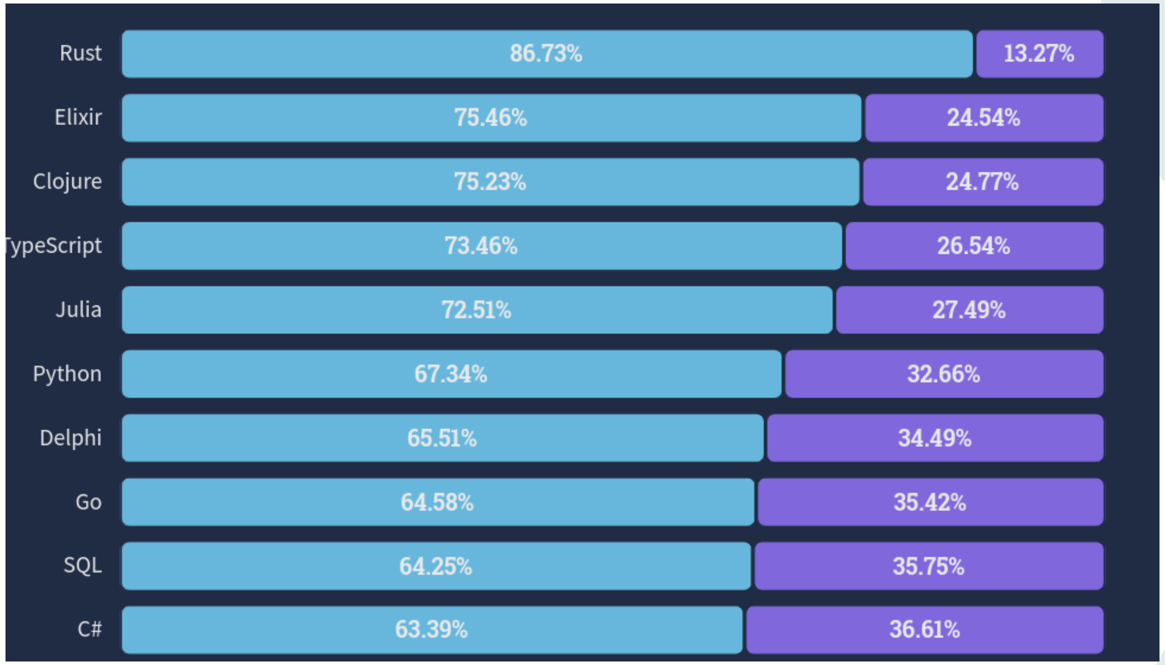
The new span will be constructed by the currently-active `Subscriber`, with the current span as its parent (if one exists

Rust is: free, accessible learning resources

- The book
- The std library reference
- Rustlings
- Rust by example
- The cargo, rustdoc, edition, … books

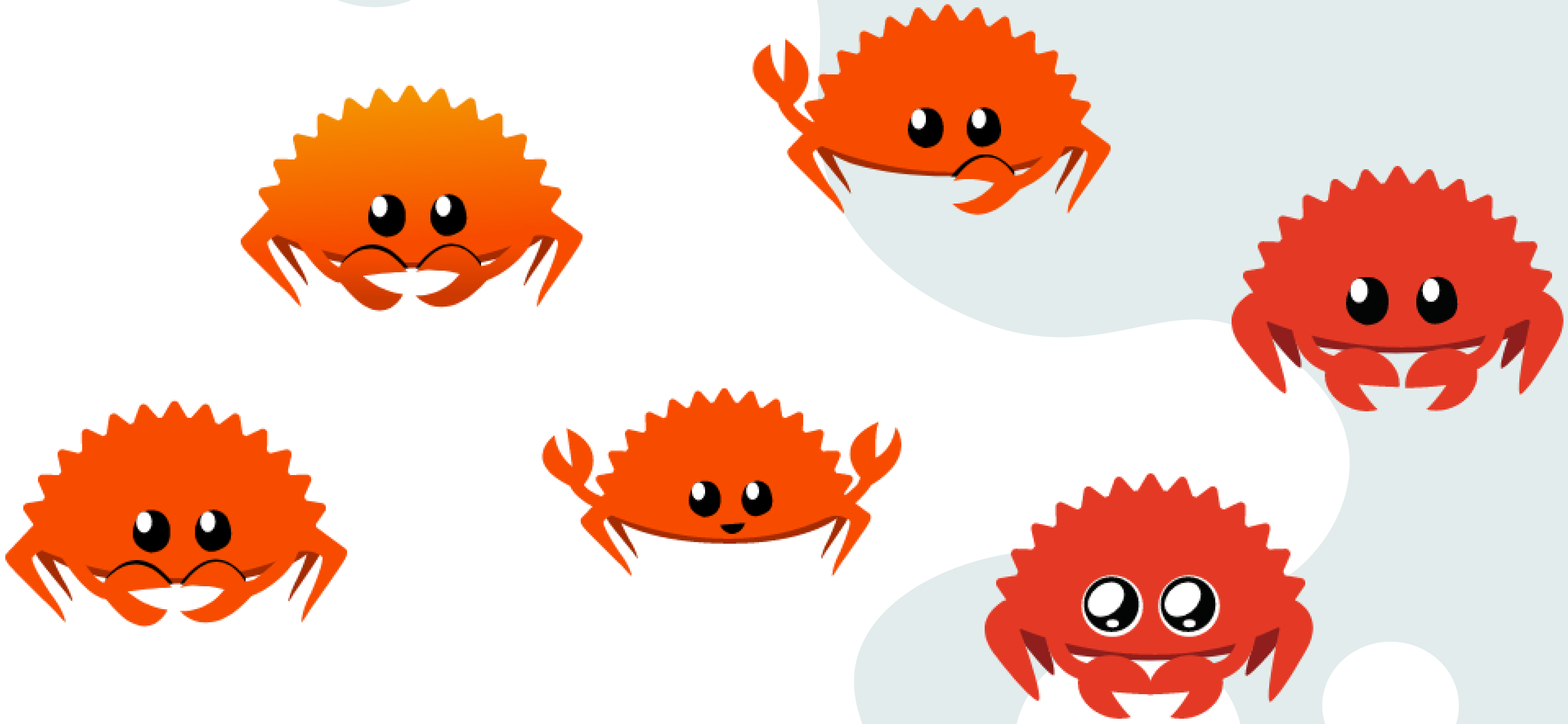
Rust is: a liked language

- 5+ years in a row, the most liked language in the Stackoverflow Developer survey



https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-language-love-dread

Rust: a production-ready language



Extra credits slide

Ferris: https://rustacean.net/