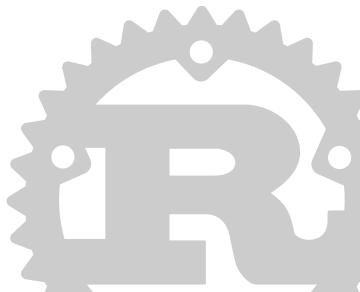


Rust<T>

Stefan Schindler (@dns2utf8)

June 27, 2016

Coredump Rapperswil

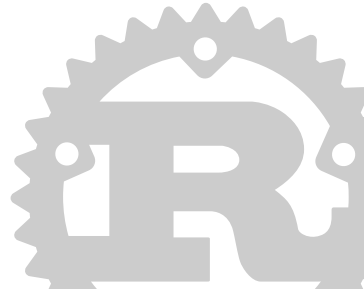


Outline

1. Admin
2. Recap form before dinner
3. Simple Generics
4. Into() complex Type
5. Enum impl
6. Transport Data with Enums
7. Search a Vector<T>
8. Sending Commands over Channels
9. Demotime
10. Questions?



Admin



- Slides are online: <https://github.com/coredump-ch/rust-t>



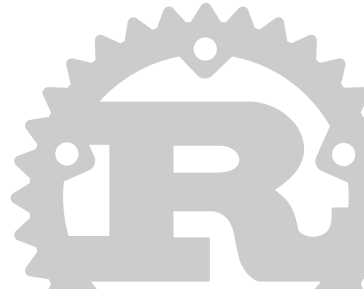
- Slides are online: <https://github.com/coredump-ch/rust-t>
- Examples are included in the **examples** directory.



- Slides are online: <https://github.com/coredump-ch/rust-t>
- Examples are included in the `examples` directory.
- Slides of Danilo & Raphael:
<https://github.com/coredump-ch/intro-to-rust>



Recap form before dinner



Example 2: Generics

```
fn min<T: Ord>(a: T, b: T) -> T {  
    if a <= b { a } else { b }  
}
```


Example 2: Generics

```
fn min<T: Ord>(a: T, b: T) -> T {  
    if a <= b { a } else { b }  
}
```

...

```
min(10i8, 20) == 10;    // T is i8  
min(10, 20u32) == 10;   // T is u32  
min("abc", "xyz") == "abc"; // Strings are Ord  
  
min(10i32, "xyz"); // error: mismatched types
```

Simple Generics



Enum

```
enum Colors {  
    Red,  
    Green,  
    Blue,  
}  
use Colors::*;  
  
fn draw(color: Colors) {  
    match color {  
        ...  
    }  
}
```

Enum

```
use Colors::*;
```

```
fn main() {  
    draw(Red);  
    draw(Blue);  
}
```

```
fn draw(color: Colors) {  
    match color {  
        Red    => 0xff0000,  
        Green  => 0x00ff00,  
        Blue   => 0x0000ff,  
    }; // no return  
}
```

Enum: non-exhaustive patterns

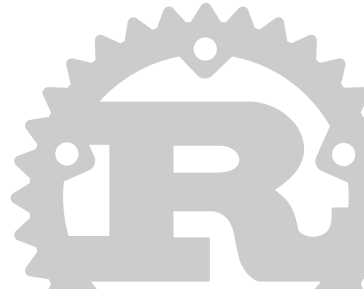
```
fn draw(color: Colors) {  
    match color {  
        Red => 0xff0000,  
        // Green => 0x00ff00,  
        Blue => 0x0000ff,  
    };  
}
```

Enum: non-exhaustive patterns

```
$ cargo run
src/main.rs:15:3: 19:4 error: non-exhaustive patterns:
↳ `Green` not covered [E0004]
src/main.rs:15   match color {
src/main.rs:16       Red => 0xff0000,
src/main.rs:17       // Green => 0x00ff00,
src/main.rs:18       Blue => 0x0000ff,
src/main.rs:19   }; // no return
src/main.rs:15:3: 19:4 help: run `rustc --explain E0004` to
↳ see a detailed explanation
error: aborting due to previous error
error: Could not compile `enum`.
```

To learn more, run the `command` again with `--verbose`.

Into() complex Type



Into() complex Type: Infrastructure

```
#[derive(Debug, Clone)]
struct MyObject {
    is : Option<isize>,
    st : Option<String>,
}

impl Into<MyObject> for isize {
    fn into(self) -> MyObject {
        MyObject {
            is : Some(self),
            st : None,
        }
    }
}
```


Into() complex Type: Infrastructure

and the implementation for `String`:

```
impl Into<MyObject> for String {  
    fn into(self) -> MyObject {  
        MyObject {  
            is : None,  
            st : Some(self),  
        }  
    }  
}
```

Into complex Type: Usage

```
let m0 = MyObject { is : Some(42), st : Some("Self  
↳ Made".into()) };
```

Into complex Type: Usage

```
let m0 = MyObject { is : Some(42), st : Some("Self  
↳ Made".into()) };
```

use with `isize`:

```
let m1 : MyObject = 23.into();
```

Into complex Type: Usage

```
let m0 = MyObject { is : Some(42), st : Some("Self  
↳ Made".into()) };
```

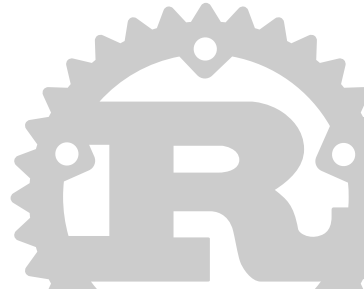
use with `isize`:

```
let m1 : MyObject = 23.into();
```

with `to_owned()` for `String`:

```
let m2 : MyObject = "Coredump.ch".to_owned().into();
```

Enum impl



Enum impl: Infrastructure

```
impl Person {  
    // A function which takes a `Person` enum as an argument  
    ↪ and  
    // returns nothing.  
    fn inspect(self) {  
        // Usage of an `enum` must cover all cases (irrefutable)  
        // so a `match` is used to branch over it.  
        match self {  
            Person::Engineer => { ... },  
            ...  
        }  
    }  
}
```

Enum impl: Usage

if we have an Enum:

```
let rohan = Person::Engineer;
```



Enum impl: Usage

if we have an Enum:

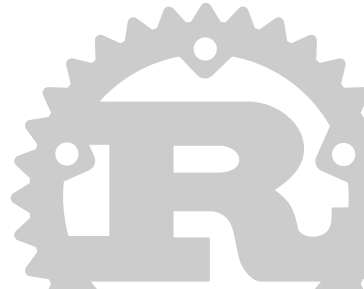
```
let rohan = Person::Engineer;
```

we can then use the method on the instance:

```
rohan.inspect();
```



Transport Data with Enums



```
#[derive(Debug)]  
enum CompoundIndex {  
    SearchIsSize( isize ),  
    SearchString( String ),  
}  
use CompoundIndex::*;
```

a number:

```
let number = SearchIsSize(42);
```



Enum Transport: Usage

a number:

```
let number = SearchIsize(42);
```

a String:

```
let string = SearchString("Coredump.ch".into());
```

Enum Transport: Usage

a number:

```
let number = SearchIsize(42);
```

a String:

```
let string = SearchString("Coredump.ch".into());
```

an empty String:

```
let string = SearchString("").into());
```

Search a Vector<T>



Search a Vector<T>: Infrastructure

```
fn find(haystack : &Vec<MyObject>, needle : &CompoundIndex)
  ↪ -> Option<MyObject> {
  for ref hay in haystack {
    match needle {
      &SearchIsize(ref needle) => {
        if let Some(ref is) = hay.is {
          if is == needle {
            return Some( (*hay).clone() );
          }
        }
      },
      ...
    } // end match
  }
  None
```

Search a Vector<T>: Infrastructure

```
fn find(haystack : &Vec<MyObject>, needle : &CompoundIndex)
  ↪ -> Option<MyObject> {
  for ref hay in haystack {
    match needle {
      ...
      &SearchString(ref needle) => {
        if let Some(ref st) = hay.st {
          if st == needle {
            return Some( (*hay).clone() );
          }
        }
      },
    } // end match
  }
  None
```


Search a Vector<T>: Usage

Prepare the Vector<MyObject>:

```
let m0 = MyObject { is : Some(42), st : Some("Self  
↳ Made".into()) };  
let m1 : MyObject = 23.into();  
let m2 : MyObject = "Coredump.ch".to_owned().into();  
  
let v = vec![m0, m1, m2];
```

Search a Vector<T>: Usage

and search it:

```
let number = SearchIsSize(42);  
println!("\n Find with number: {:?} => {:?}", number,  
    ↪ find(&v, &number));
```

```
let string = SearchString("").into();  
println!("\n Find with String: {:?} => {:?}", string,  
    ↪ find(&v, &string));
```

```
let string = SearchString("Coredump.ch".into());  
println!("\n Find with String: {:?} => {:?}", string,  
    ↪ find(&v, &string));
```

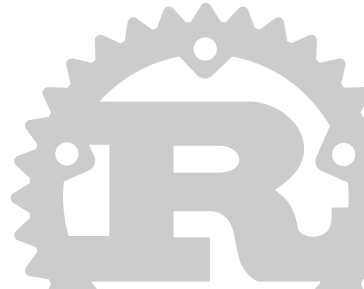
Search a Vector<T>: Output

```
Find with number: SearchIsize(42) => Some(MyObject { is:  
↪ Some(42), st: Some("Self Made") })
```

```
Find with String: SearchString("") => None
```

```
Find with String: SearchString("Coredump.ch") =>  
↪ Some(MyObject { is: None, st: Some("Coredump.ch") })
```

Sending Commands over Channels



Sending Commands over Channels

Infrastructure:

```
use std::sync::mpsc::channel;
```

```
let (tx, rx) = channel();
```



Sending Commands over Channels

Infrastructure:

```
use std::sync::mpsc::channel;
```

```
let (tx, rx) = channel();
```

Usage:

```
tx.send(42).unwrap();  
assert_eq!(42, rx.recv().unwrap());
```

Sending Commands over Channels

Infrastructure:

```
use std::sync::mpsc::channel;
```

```
let (tx, rx) = channel();
```

Usage:

```
tx.send(42).unwrap();  
assert_eq!(42, rx.recv().unwrap());
```

Works with complex Types:

```
let (tx, rx) = channel::<MyCommands<u64>>>();
```

Massive errors

Natural occurrences:

```
let n = 10;  
let y = (["a", "b"])[n]; // panics  
  
my_io_function().unwrap() // maybe panics
```


Massive errors

Natural occurrences:

```
let n = 10;  
let y = (["a", "b"])[n]; // panics  
  
my_io_function().unwrap() // maybe panics
```

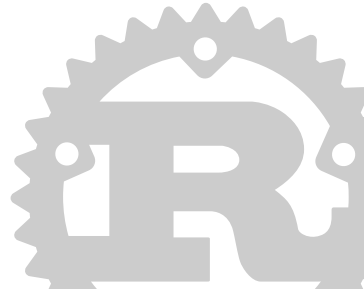
Simulated:

```
panic!("with a message")
```

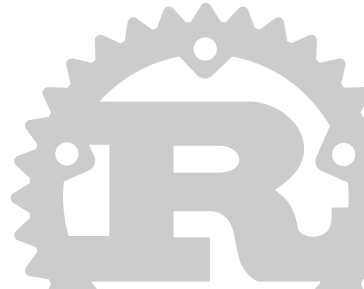
```
let pool = ThreadPool::new(4);

let rx = {
    let (tx, rx) = channel();
    for i in 0..8 {
        let tx = tx.clone();
        pool.execute(move || {
            if i == 4 {panic!("unexpected panic");} // --
            ↪ unexpected failure added here --
            tx.send(i).unwrap();
        });
    }
    rx
};
```

Demotime



Questions?



Thank you!

www.coredump.ch

