Rust and Haskell

sitting in a tree

Lisa 'lislis' Passing

Lambda.World Cádiz 2018

$ whoami

$ whoami

Hi, I'm Lisa 👌

$ whoami

Hi, I'm Lisa 👌

Web dev, game jammer, wannabe digital artist, fp enthusiast

$ whoami

Hi, I'm Lisa 👌

Web dev, game jammer, wannabe digital artist, fp enthusiast

From a dynamically typed language background (JS, Ruby)

$ whoami

Hi, I'm Lisa 👌

Web dev, game jammer, wannabe digital artist, fp enthusiast

From a dynamically typed language background (JS, Ruby)

Came to FP through Clojure/Script

$ whoami

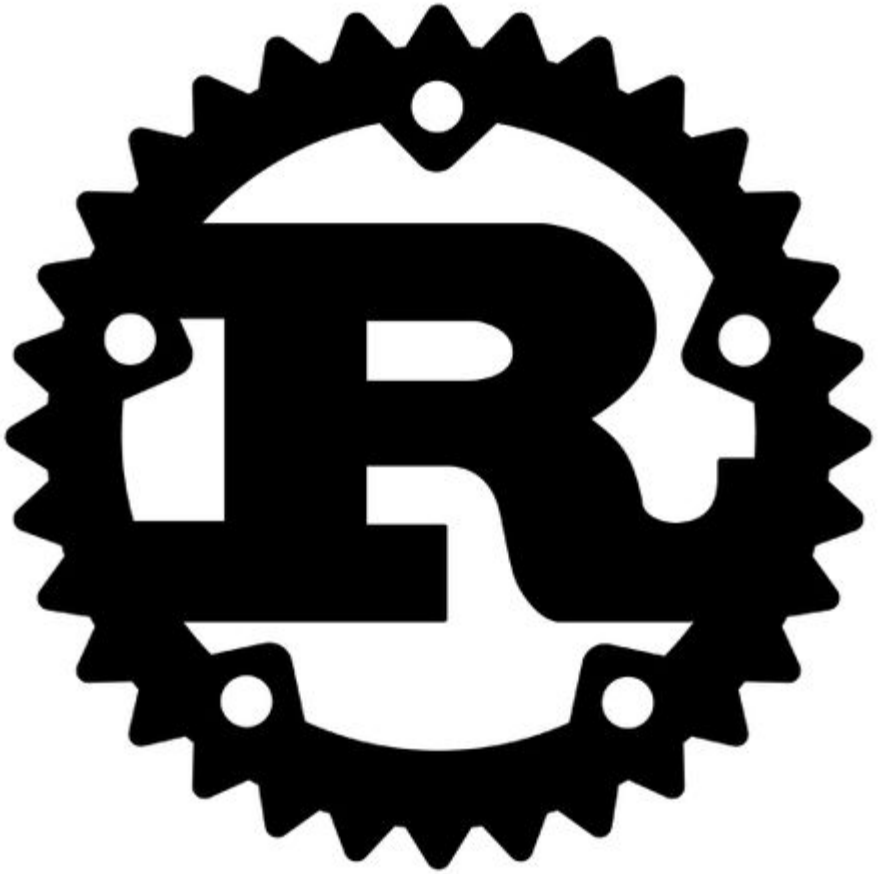
Hi, I'm Lisa 👌

Web dev, game jammer, wannabe digital artist, fp enthusiast

From a dynamically typed language background (JS, Ruby)

Came to FP through Clojure/Script

Started learning Rust 1.5 years ago



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

Documentation Install Community Contribute

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Install Rust **1.29.1**

September 25, 2018

See who's using Rust, and read more about Rust in production.

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

Run

```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                     "Ciao", "こんにちは", "안녕하세요",
                     "Cześć", "Olá", "Здравствуйте",
                     "Chào bạn", "您好", "Hallo",
                     "Hej", "Ahoj", "سلام","สวัสดี"];

    for (num, greeting) in greetings.iter().enumerate() {
        print!("{} : ", greeting);
        match num {
            0 =>  println!("This code is editable and runnal
            1 =>  println!("¡Este código es editable y ejecu
            2 =>  println!("Ce code est modifiable et exécut
            3 =>  println!("Questo codice è modificabile ed
            4 =>  println!("このコードは編集して実行出来ます
            5 =>  println!("여기에서 코드를 수정하고 실행할
            6 =>  println!("Ten kod można edytować oraz uruc
            7 =>  println!("Este código é editável e executa
            8 =>  println!("Этот код можно отредактировать
            9 =>  println!("Bạn có thể edit và run code trực
            10 => println!("这段代码是可以编辑并且能够运行的
            11 => println!("Dieser Code kann bearbeitet und
            12 => println!("Den här koden kan redigeras och
            13 => println!("Tento kód můžete upravit a spust
            14 => println!("این کد قابلیت ویرایش و اجرا دارد
            15 => println!("โค้ดนี้สามารถแก้ไขได้และรันได้"),
            _ =>  {},
        }
    }
}
```

More examples

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

Documentation Install Community Contribute

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Install Rust **1.29.1**

September 25, 2018

See who's using Rust, and read more about Rust in production.

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

More examples

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

Documentation Install Community Contribute

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Install Rust **1.29.1**

September 25, 2018

See who's using Rust, and read more about Rust in production.

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

```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                     "Ciao", "こんにちは", "안녕하세요",
                     "Cześć", "Olá", "Здравствуйте",
                     "Chào bạn", "您好", "Hallo",
                     "Hej", "Ahoj", "سلام","สวัสดี"];

    for (num, greeting) in greetings.iter().enumerate() {
        print!("{} : ", greeting);
        match num {
            0 =>  println!("This code is editable and runnal
            1 =>  println!("¡Este código es editable y ejecu
            2 =>  println!("Ce code est modifiable et exécut
            3 =>  println!("Questo codice è modificabile ed
            4 =>  println!("このコードは編集して実行出来ます
            5 =>  println!("여기에서 코드를 수정하고 실행할
            6 =>  println!("Ten kod można edytować oraz uruc
            7 =>  println!("Este código é editável e executa
            8 =>  println!("Этот код можно отредактировать
            9 =>  println!("Bạn có thể edit và run code trực
            10 => println!("这段代码是可以编辑并且能够运行的
            11 => println!("Dieser Code kann bearbeitet und
            12 => println!("Den här koden kan redigeras och
            13 => println!("Tento kód můžete upravit a spust
            14 => println!("این کد قابلیت ویرایش و اجرا دارد
            15 => println!("โค้ดนี้สามารถแก้ไขได้และรันได้"),
            _ =>  {},
        }
    }
}
```

Run

More examples

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

Documentation Install Community Contribute

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Install Rust **1.29.1**

September 25, 2018

See who's using Rust, and read more about Rust in production.

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

Run

```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                     "Ciao", "こんにちは", "안녕하세요",
                     "Cześć", "Olá", "Здравствуйте",
                     "Chào bạn", "您好", "Hallo",
                     "Hej", "Ahoj", "سلام","สวัสดี"];

    for (num, greeting) in greetings.iter().enumerate() {
        print!("{} : ", greeting);
        match num {
            0 =>  println!("This code is editable and runnal
            1 =>  println!("¡Este código es editable y ejecu
            2 =>  println!("Ce code est modifiable et exécut
            3 =>  println!("Questo codice è modificabile ed
            4 =>  println!("このコードは編集して実行出来ます
            5 =>  println!("여기에서 코드를 수정하고 실행할
            6 =>  println!("Ten kod można edytować oraz uruc
            7 =>  println!("Este código é editável e executa
            8 =>  println!("Этот код можно отредактировать
            9 =>  println!("Bạn có thể edit và run code trực
            10 => println!("这段代码是可以编辑并且能够运行的
            11 => println!("Dieser Code kann bearbeitet und
            12 => println!("Den här koden kan redigeras och
            13 => println!("Tento kód můžete upravit a spust
            14 => println!("این کد قابلیت ویرایش و اجرا دارد
            15 => println!("โค้ดนี้สามารถแก้ไขได้และรันได้"),
            _ =>  {},
        }
    }
}
```

More examples

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

Documentation Install Community Contribute

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Install Rust **1.29.1**

September 25, 2018

See who's using Rust, and read more about Rust in production.

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

```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                     "Ciao", "こんにちは", "안녕하세요",
                     "Cześć", "Olá", "Здравствуйте",
                     "Chào bạn", "您好", "Hallo",
                     "Hej", "Ahoj", "سلام","สวัสดี"];

    for (num, greeting) in greetings.iter().enumerate() {
        print!("{} : ", greeting);
        match num {
            0 =>  println!("This code is editable and runnal
            1 =>  println!("¡Este código es editable y ejecu
            2 =>  println!("Ce code est modifiable et exécut
            3 =>  println!("Questo codice è modificabile ed
            4 =>  println!("このコードは編集して実行出来ます
            5 =>  println!("여기에서 코드를 수정하고 실행할
            6 =>  println!("Ten kod można edytować oraz uruc
            7 =>  println!("Este código é editável e executa
            8 =>  println!("Этот код можно отредактировать
            9 =>  println!("Bạn có thể edit và run code trực
            10 => println!("这段代码是可以编辑并且能够运行的
            11 => println!("Dieser Code kann bearbeitet und
            12 => println!("Den här koden kan redigeras och
            13 => println!("Tento kód můžete upravit a spust
            14 => println!("این کد قابلیت ویرایش و اجرا دارد
            15 => println!("โค้ดนี้สามารถแก้ไขได้และรันได้"),
            _ =>  {},
        }
    }
}
```

Run

More examples

Let's see

Let's see

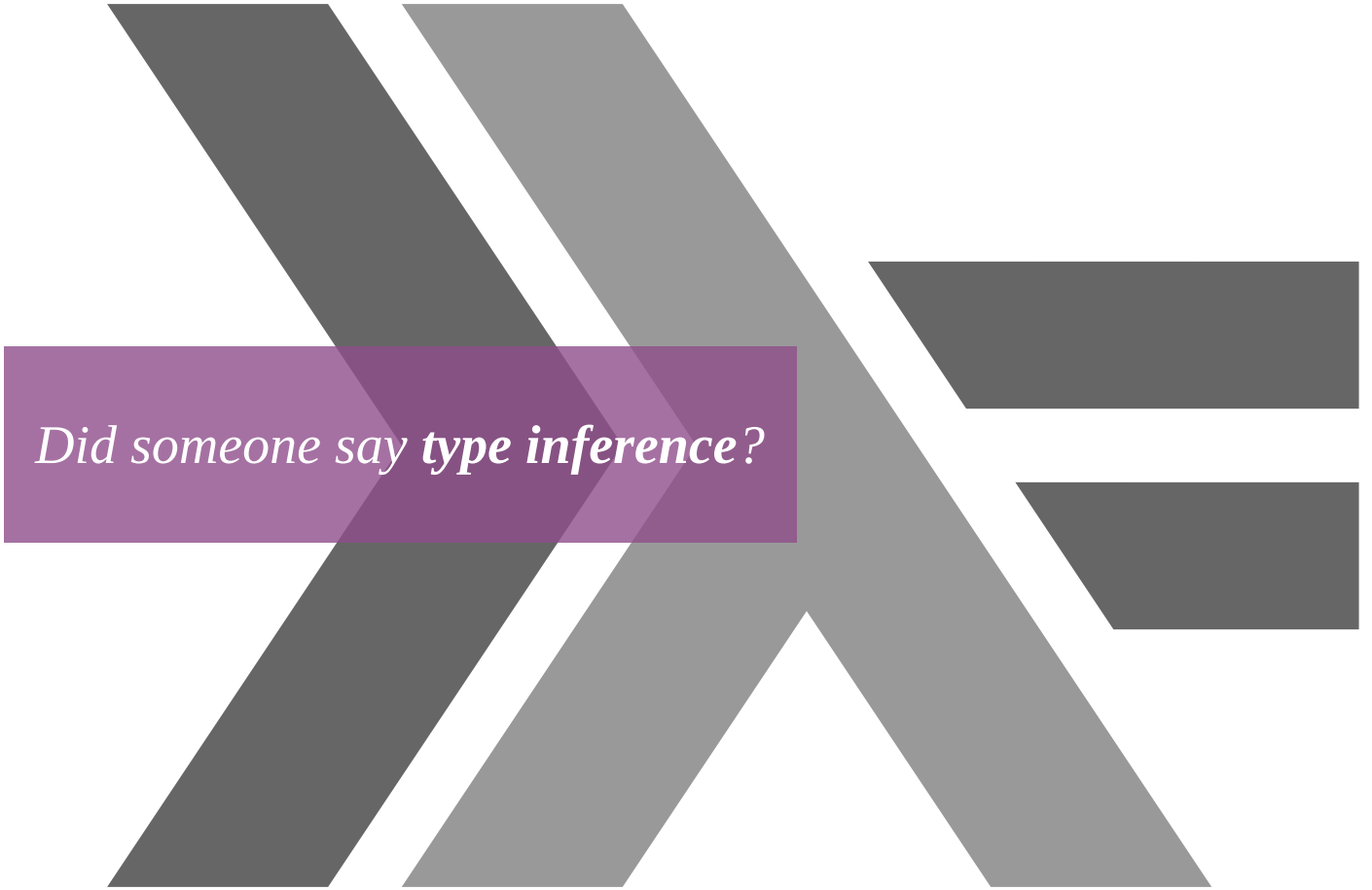
- type inference
- pattern matching
- trait-based generics
- zero-cost abstractions

Let's see

- type inference
- pattern matching
- trait-based generics
- zero-cost abstractions

Could make a pretty good fp talk 

Type inference


*Did someone say **type inference**?*

Type inference in Haskell

Type inference in Haskell

```
doubleMe x = x * 2
```

Type inference in Haskell

```
doubleMe x = x * 2

doubleMe 8
```

```
16
```

Type inference in Haskell

```
doubleMe x = x * 2

doubleMe 8
```

```
16
```

Haskell infers that this can only work with numbers.

Type inference in Haskell

```
doubleMe x = x * 2

doubleMe 8
```

```
16
```

Haskell infers that this can only work with numbers.

```
doubleMe "this is a trick!"

doubleMe []
```

Type inference in Haskell

```
doubleMe x = x * 2

doubleMe 8
```

```
16
```

Haskell infers that this can only work with numbers.

```
doubleMe "this is a trick!"

doubleMe []
```

```
ERROR!
```

Type inference in Haskell

```
doubleMe x = x * 2

doubleMe 8
```

```
16
```

Haskell infers that this can only work with numbers.

```
doubleMe "this is a trick!"

doubleMe []
```

```
ERROR!
```

We get an error for everything else.

Type inference in Haskell

This works with lists, too!

Type inference in Haskell

This works with lists, too!

```
doubleFirstOfList xs = head xs * 2
```

Type inference in Haskell

This works with lists, too!

```
doubleFirstOfList xs = head xs * 2
```

```
doubleFirstOfList [239, 3482, 23, 23]
```

Type inference in Haskell

This works with lists, too!

```
doubleFirstOfList xs = head xs * 2
```

```
doubleFirstOfList [239, 3482, 23, 23]
```

```
478
```

Type inference in Haskell

We don't have to annotate types,

but it's nice when we do.

Type inference in Haskell

We don't have to annotate types,

but it's nice when we do.

```
doubleMe' Int -> Int
doubleMe' x = x * 2
```

Type inference in Haskell

We don't have to annotate types,

but it's nice when we do.

```
doubleMe' Int -> Int
doubleMe' x = x * 2
```

```
doubleFirstOfList' :: [Int] -> Int
doubleFirstOfList' xs = head xs * 2
```

What about Rust?

Type inference in Rust

Type inference in Rust

- Types are inferred when possible

Type inference in Rust

- Types are inferred when possible

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

  let doubleElem = elem * 2;

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

Type inference in Rust

- Types are inferred when possible

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

  let doubleElem = elem * 2;

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

```
10
```

Type inference in Rust

```
fn main() {
  let vector = vec![2, 16, 348];

  let double_first_of_vec = vector[0] * 2;

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

Type inference in Rust

```
fn main() {
  let vector = vec![2, 16, 348];

  let double_first_of_vec = vector[0] * 2;

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

```
4
```

Type inference in Rust

Function params and return values

have to always be annotated!

Type inference in Rust

Function params and return values

have to always be annotated!

```
fn double_first_of_vec(v:Vec<u32>) -> u32 {
  v[0] * 2
}

fn main() {
  let vector = vec![2, 16, 348];
  println!("{}", double_first_of_vec(vector));
}
```

Type inference in Rust

Function params and return values

have to always be annotated!

```
fn double_first_of_vec(v:Vec<u32>) -> u32 {
  v[0] * 2
}

fn main() {
  let vector = vec![2, 16, 348];
  println!("{}", double_first_of_vec(vector));
}
```

```
4
```

Trait-based generics

What are generics in Rust?

What are generics in Rust?

An abstract stand-in for a concrete type

What are generics in Rust?

An abstract stand-in for a concrete type

```
fn first_and_last<T>(v: &Vec<T>) -> Vec<&T> {

}
```

What are generics in Rust?

An abstract stand-in for a concrete type

```
fn first_and_last<T>(v: &Vec<T>) -> Vec<&T> {
  vec![v.first().unwrap(),
       v.iter().last().unwrap()]
}
```

What are generics in Rust?

An abstract stand-in for a concrete type

```
fn first_and_last<T>(v: &Vec<T>) -> Vec<&T> {
  vec![v.first().unwrap(),
       v.iter().last().unwrap()]
}
```

```
fn main() {
  let nums = vec![2, 16, 348];
  println!("{:?}", first_and_last(&nums));

  let strings = vec!["Hello", "a", "foo", "World"];
  println!("{:?}", first_and_last(&strings));
}
```

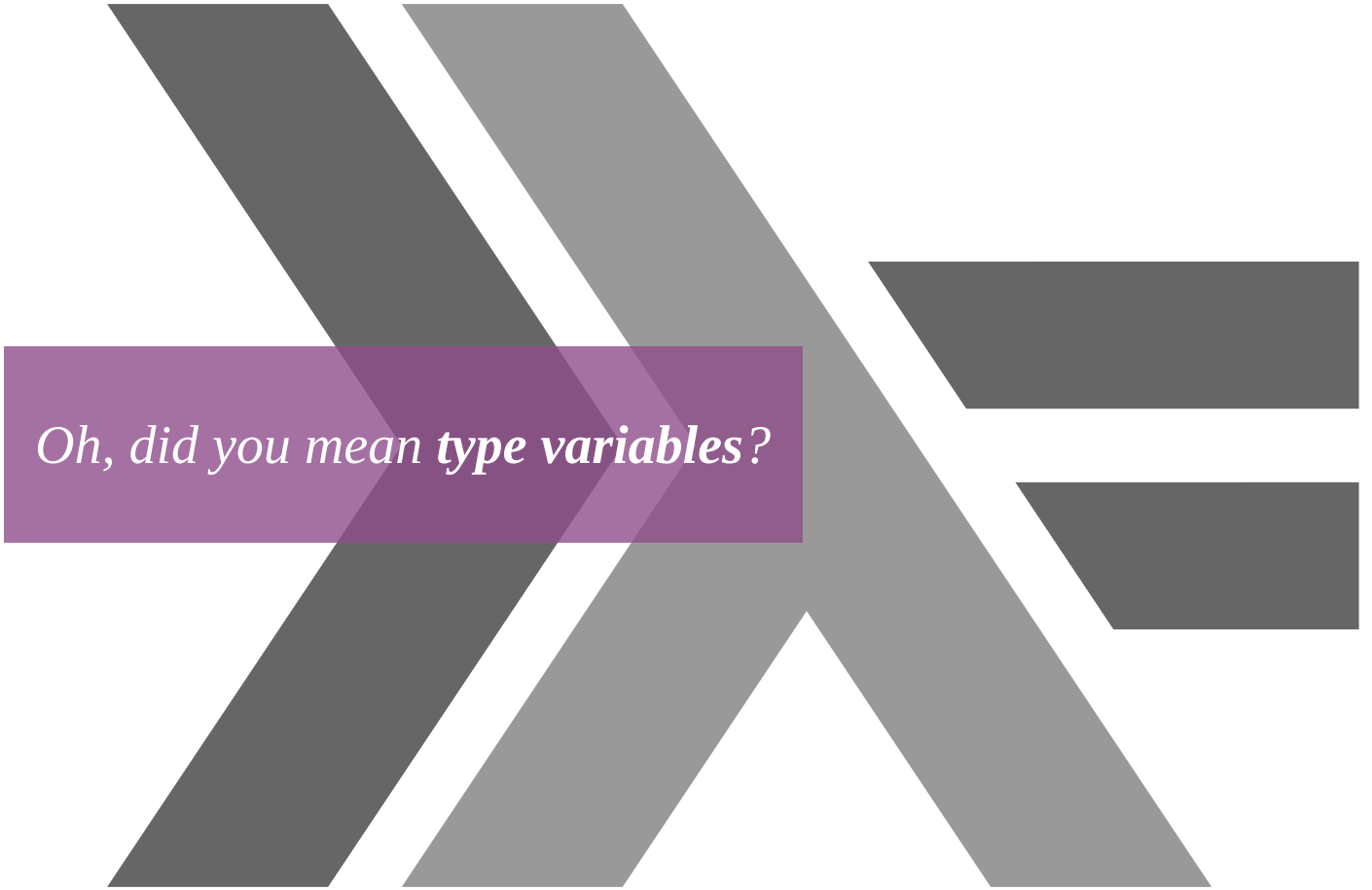
What are generics in Rust?

An abstract stand-in for a concrete type

```
fn first_and_last<T>(v: &Vec<T>) -> Vec<&T> {
  vec![v.first().unwrap(),
       v.iter().last().unwrap()]
}
```

```
fn main() {
  let nums = vec![2, 16, 348];
  println!("{:?}", first_and_last(&nums));

  let strings = vec!["Hello", "a", "foo", "World"];
  println!("{:?}", first_and_last(&strings));
}
```

```
[2, 348]
["Hello", "World"]
```


*Oh, did you mean **type variables**?*

Type variables in Haskell

Type variables in Haskell

```
firstAndLast :: [a] -> [a]
firstAndLast xs = [head xs, last xs]
```

Type variables in Haskell

```
firstAndLast :: [a] -> [a]
firstAndLast xs = [head xs, last xs]
```

```
firstAndLast [45, 28, 645, 23]
```

```
[45, 23]
```

What are Traits in Rust?

What are Traits in Rust?

Traits define behaviour that Types can implement.

What are Traits in Rust?

Traits define behaviour that Types can implement.

Example `Display` Trait for user facing string output.

https://doc.rust-lang.org/std/fmt/trait.Display.html

Click or press 'S' to search, '?' for more options...

Trait std::fmt::Display

1.0.0 [−][src]

[+] Show declaration

[−] Format trait for an empty format, `{}`.

`Display` is similar to `Debug`, but `Display` is for user-facing output, and so cannot be derived.

For more information on formatters, see the module-level documentation.

Examples

Implementing `Display` on a type:

```
use std::fmt;

struct Point {
    x: i32,
    y: i32,
}

impl fmt::Display for Point {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "({}, {})", self.x, self.y)
    }
}

let origin = Point { x: 0, y: 0 };

println!("The origin is: {}", origin);
```

Run

Trait Display

Required Methods

fmt

Implementations on Foreign Types

Utf8Lossy

TryFromSliceError

Implementors

std::fmt

Structs

Arguments
DebugList
DebugMap
DebugSet
DebugStruct
DebugTuple
Error
Formatter

Implementing `Display` on a type

Implementing `Display` on a type

```
use std::fmt;

struct Point {
  x: i32,
  y: i32
}
```

Implementing `Display` on a type

```
use std::fmt;

struct Point {
  x: i32,
  y: i32
}

impl fmt::Display for Point {
  fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
    write!(f, "({}, {})", self.x, self.y)
  }
}
```

Implementing `Display` on a type

```
use std::fmt;

struct Point {
  x: i32,
  y: i32
}

impl fmt::Display for Point {
  fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
    write!(f, "({}, {})", self.x, self.y)
  }
}

let p = Point { x: 0, y: 0 };

println!("The point is: {}", p);
```

Implementing `Display` on a type

```
use std::fmt;

struct Point {
  x: i32,
  y: i32
}

impl fmt::Display for Point {
  fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
    write!(f, "({}, {})", self.x, self.y)
  }
}

let p = Point { x: 0, y: 0 };

println!("The point is: {}", p);
```

```
The point is: (0, 0)
```


Sounds a lot like ***type classes***

Type classes in Haskell

Type classes in Haskell

Let's look at the `Show` type class

Type classes in Haskell

Let's look at the `Show` type class

```
data Point = Point { x :: Int
                   , y :: Int
                   } deriving (Show)
```

Type classes in Haskell

Let's look at the `Show` type class

```
data Point = Point { x :: Int
                   , y :: Int
                   } deriving (Show)
```

```
let p1 = Point 2 3

show p1
```

Type classes in Haskell

Let's look at the `Show` type class

```
data Point = Point { x :: Int
                   , y :: Int
                   } deriving (Show)
```

```
let p1 = Point 2 3

show p1
```

```
Point {x = 2, y = 3}
```

Alternatively we can make a function

```
display :: Point -> String
display (Point {x=x, y=y}) = "The point is (" ++ show x ++
                             "," ++ show y ++")"
```

Alternatively we can make a function

```
display :: Point -> String
display (Point {x=x, y=y}) = "The point is (" ++ show x ++
                             "," ++ show y ++")"
```

```
let p = Point 0 0
display p
```

Alternatively we can make a function

```
display :: Point -> String
display (Point {x=x, y=y}) = "The point is (" ++ show x ++
                             "," ++ show y ++")"
```

```
let p = Point 0 0
display p
```

```
The point is (0, 0)
```

Trait bounds!

Trait bounds!

```rust
// Rust
use std::fmt::Display;

fn shout_out<T: Display>(s: T) -> String {
  format!("{}!!!!!!!!!", s)
}

fn main() {
  println!("{}", shout_out("Uuuuh"));
}
```

Trait bounds!

```rust
// Rust
use std::fmt::Display;

fn shout_out<T: Display>(s: T) -> String {
  format!("{}!!!!!!!!!", s)
}

fn main() {
  println!("{}", shout_out("Uuuuh"));
}
```

Class constraints!

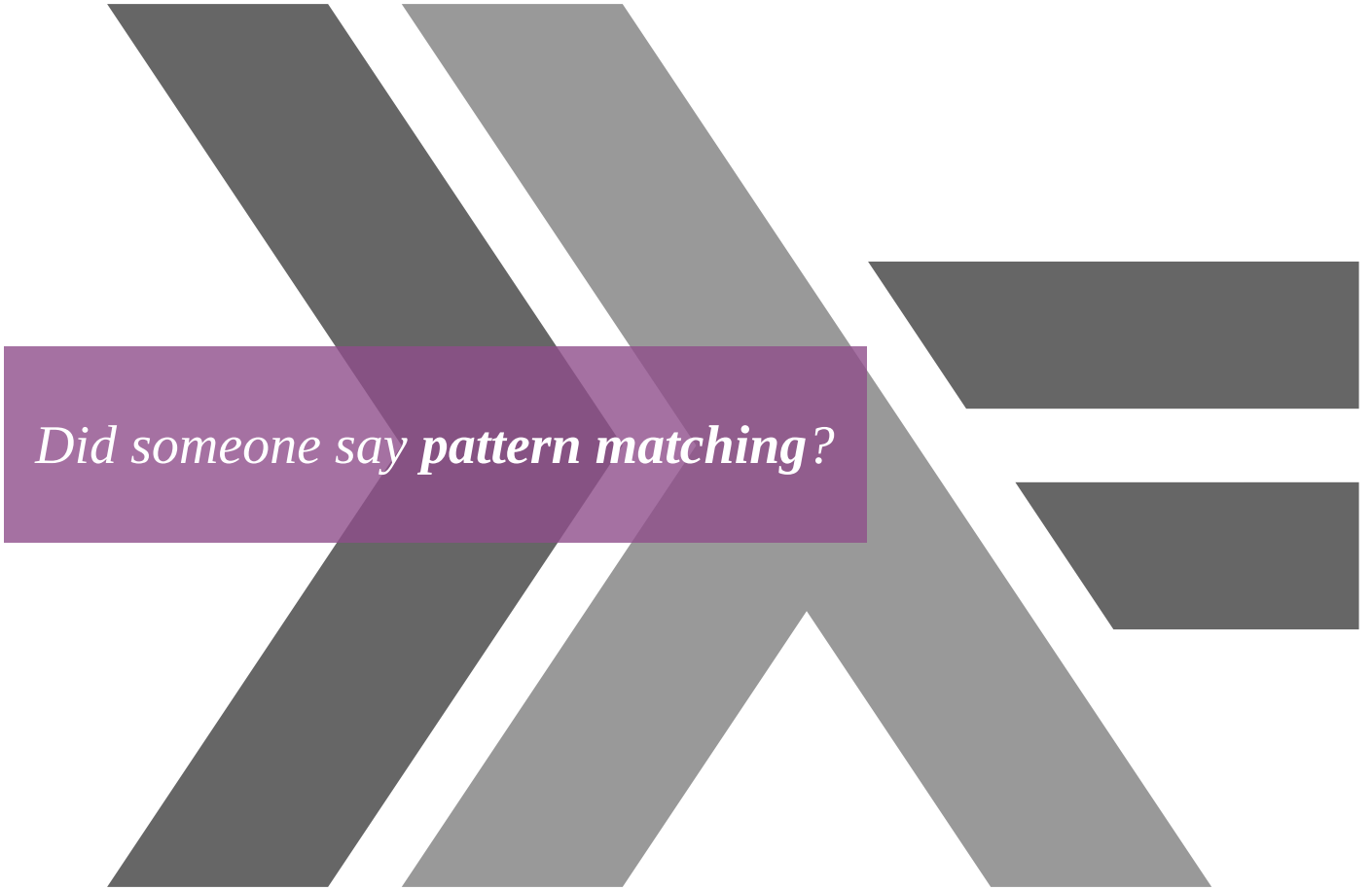
Trait bounds!

```rust
// Rust
use std::fmt::Display;

fn shout_out<T: Display>(s: T) -> String {
  format!("{}!!!!!!!!!!", s)
}

fn main() {
  println!("{}", shout_out("Uuuuh"));
}
```

Class constraints!

```haskell
-- Haskell
shoutOut :: (Show a) => a -> String
shoutOut x = (show x) ++ "!!!!!!!!!!"

shoutOut "Yeah"
```

Pattern matching


*Did someone say **pattern matching**?*

Pattern matching in Haskell

Pattern matching in Haskell

```haskell
data Shape = Circle Float Float Float | Rectangle Float Float Float F

surface :: Shape -> Float
surface (Circle _ _ r) = pi * r ^ 2
surface (Rectangle x1 y1 x2 y2) = (abs $ x2 - x1)
                                  * (abs $ y2 - y1)
```

Pattern matching in Haskell

```
data Shape = Circle Float Float Float | Rectangle Float Float Float F

surface :: Shape -> Float
surface (Circle _ _ r) = pi * r ^ 2
surface (Rectangle x1 y1 x2 y2) = (abs $ x2 - x1)
                                  * (abs $ y2 - y1)
```

```
surface $ Circle 10 10 10
```

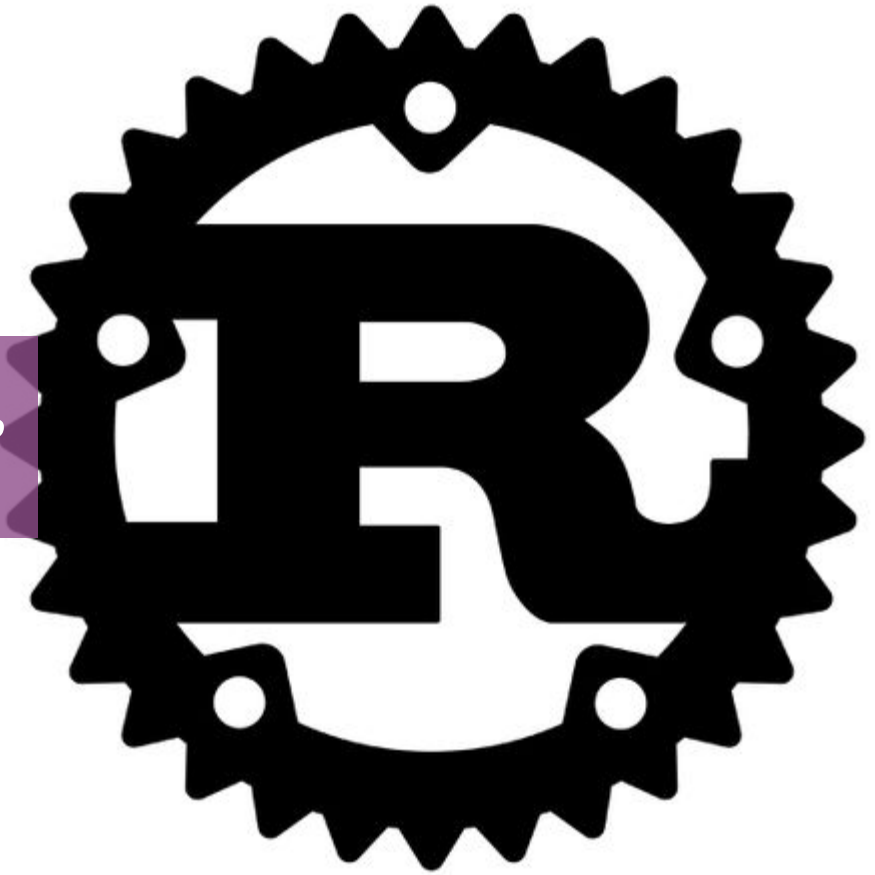
Pattern matching in Haskell

```
data Shape = Circle Float Float Float | Rectangle Float Float Float F

surface :: Shape -> Float
surface (Circle _ _ r) = pi * r ^ 2
surface (Rectangle x1 y1 x2 y2) = (abs $ x2 - x1)
                                  * (abs $ y2 - y1)
```

```
surface $ Circle 10 10 10
```

```
314.15927
```

And in Rust?



Pattern matching in Rust

Pattern matching in Rust

```
use std::f32::consts::PI;

struct Circle(f32, f32, f32);
struct Rectangle(f32, f32, f32, f32);

enum Shape {
  Circle(f32, f32, f32),
  Rectangle(f32, f32, f32, f32)
}
```

Pattern matching in Rust

```
use std::f32::consts::PI;

struct Circle(f32, f32, f32);
struct Rectangle(f32, f32, f32, f32);

enum Shape {
  Circle(f32, f32, f32),
  Rectangle(f32, f32, f32, f32)
}

fn surface(s: Shape) -> f32 {
  match s {
    Shape::Circle(_, _, r) => ( PI * r.powf(2.0)  ),
    Shape::Rectangle(x1, y1, x2, y2) => {
      (x2 - x1).abs() * (y2 - y1).abs()
    },
  }
}
```

Pattern matching in Rust

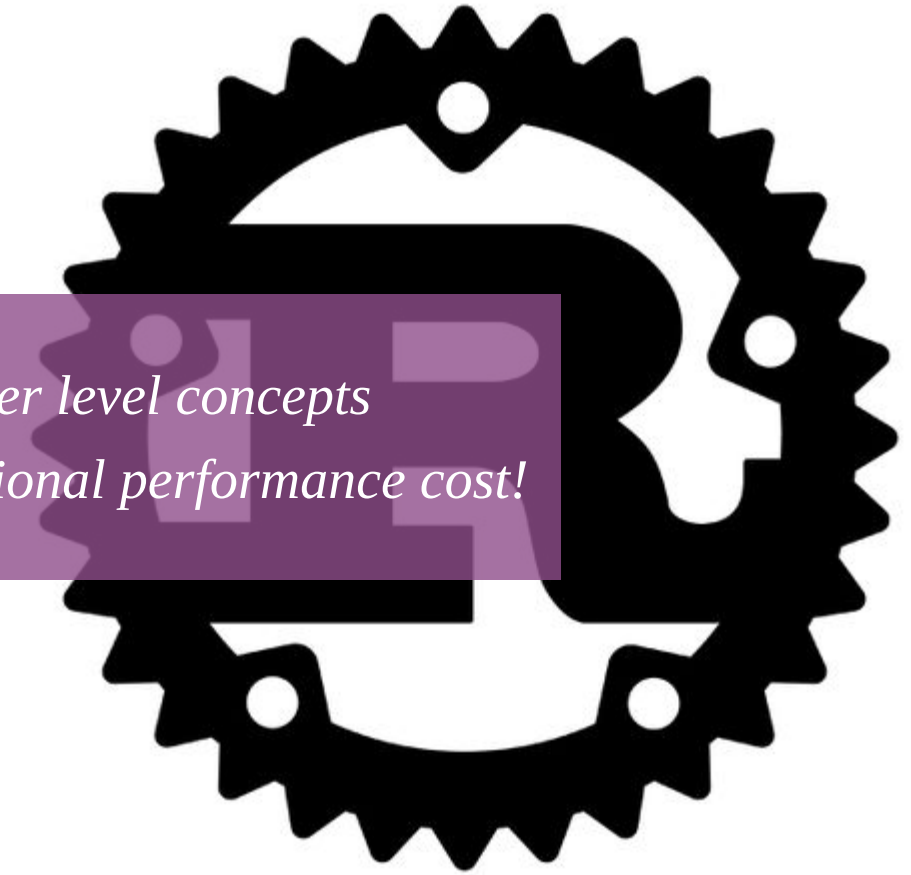
```
fn main() {
  let circle = Shape::Circle(10.0, 10.0, 10.0);
  println!("{:?}", surface(circle));
}
```

Pattern matching in Rust

```
fn main() {
  let circle = Shape::Circle(10.0, 10.0, 10.0);
  println!("{:?}", surface(circle));
}
```

```
314.15927
```

Zero-cost abstractions



Using higher level concepts
at no additional performance cost!

Example

Higher order functions

Higher order functions

in Haskell

Higher order functions

in Haskell

Like map

Higher order functions

in Haskell

Like map

```
map (+3) [1,5,3,1,6]
```

Higher order functions

in Haskell

Like map

```
map (+3) [1,5,3,1,6]
```

```
[4,8,6,4,9]
```

Higher order functions

in Haskell

Like map

```
map (+3) [1,5,3,1,6]
```

```
[4,8,6,4,9]
```

or filter

Higher order functions

in Haskell

Like map

```
map (+3) [1,5,3,1,6]
```

```
[4,8,6,4,9]
```

or filter

```
filter (>3) [1,5,3,2,1,6,4,3,2,1]
```

Higher order functions

in Haskell

Like map

```
map (+3) [1,5,3,1,6]
```

```
[4,8,6,4,9]
```

or filter

```
filter (>3) [1,5,3,2,1,6,4,3,2,1]
```

```
[5,6,4]
```

What about Rust?

About functions in Rust (pt 1)

About functions in Rust (pt 1)

Function declarations in Rust are statements.

Function calls are expressions.

About functions in Rust (pt 1)

Function declarations in Rust are statements.

Function calls are expressions.

Statements cannot be evaluated and therefore
not be assigned to variables
nor used as an arbitrary value.

About functions in Rust (pt 1)

Function declarations in Rust are statements.

Function calls are expressions.

Statements cannot be evaluated and therefore
not be assigned to variables
nor used as an arbitrary value.

Expressions can be.

About functions in Rust (pt 2)

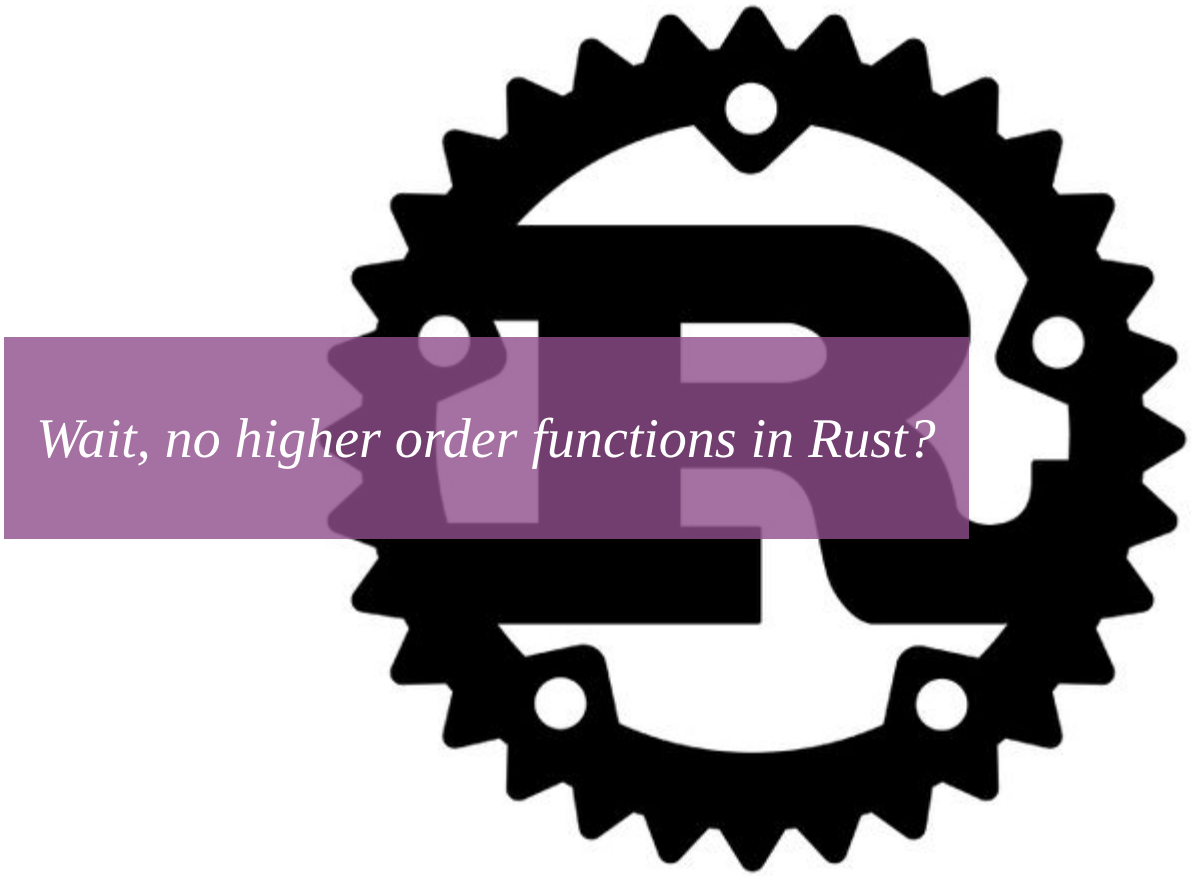
About functions in Rust (pt 2)

A function's name becomes part of its type,
therefore two functions with the same signature are still different.

About functions in Rust (pt 2)

A function's name becomes part of its type,
therefore two functions with the same signature are still different.

Dynamically creating functions
and passing or returning them is
not (easily) possible because types need to annotated beforehand.


Wait, no higher order functions in Rust?

Wait, no higher order functions in Rust?

Well, not like this...

But there is a another way!

Closures in Rust

Closures in Rust

Closures are anonymous functions that can capture their environment.

Closures in Rust

Closures are anonymous functions that can capture their environment.

Their declarations can be stored in variables.

Closures in Rust

Closures are anonymous functions that can capture their environment.

Their declarations can be stored in variables.

They don't need type annotations for params or return values (Types are inferred on first use).

Closures in Rust

Closures are anonymous functions that can capture their environment.

Their declarations can be stored in variables.

They don't need type annotations for params or return values (Types are inferred on first use).

Every closure has its own type but implements one of the traits `Fn`, `FnMut`, `FnOnce`. This can be used with trait bounds when defining structs that hold closures.

Using closures with `Iterator`s

Using closures with `Iterator`s

```
fn main() {
  let vmap: Vec<i32> = vec![1, 5, 3, 1, 6]
      .into_iter()
      .map(|x| x + 3)
      .collect();
}
```

Using closures with `Iterator`s

```
fn main() {
  let vmap: Vec<i32> = vec![1, 5, 3, 1, 6]
      .into_iter()
      .map(|x| x + 3)
      .collect();

  let vfilter: Vec<i32> = vec![1,5,3,2,1,6,4,3,2,1]
      .into_iter()
      .filter(|x| x > &3)
      .collect();

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

Using closures with `Iterator`s

```
fn main() {
  let vmap: Vec<i32> = vec![1, 5, 3, 1, 6]
      .into_iter()
      .map(|x| x + 3)
      .collect();

  let vfilter: Vec<i32> = vec![1,5,3,2,1,6,4,3,2,1]
      .into_iter()
      .filter(|x| x > &3)
      .collect();

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

```
[4, 8, 6, 4, 9]
[5, 6, 4]
```

Zero-cost abstractions

Using higher level concepts
at no additional performance cost!

phew

phew

That was a lot.

Rust and Haskell

Rust and Haskell

Rust is not a functional language,

but it has learned a lot of good things

from Haskell and the functional world.

More nice things!

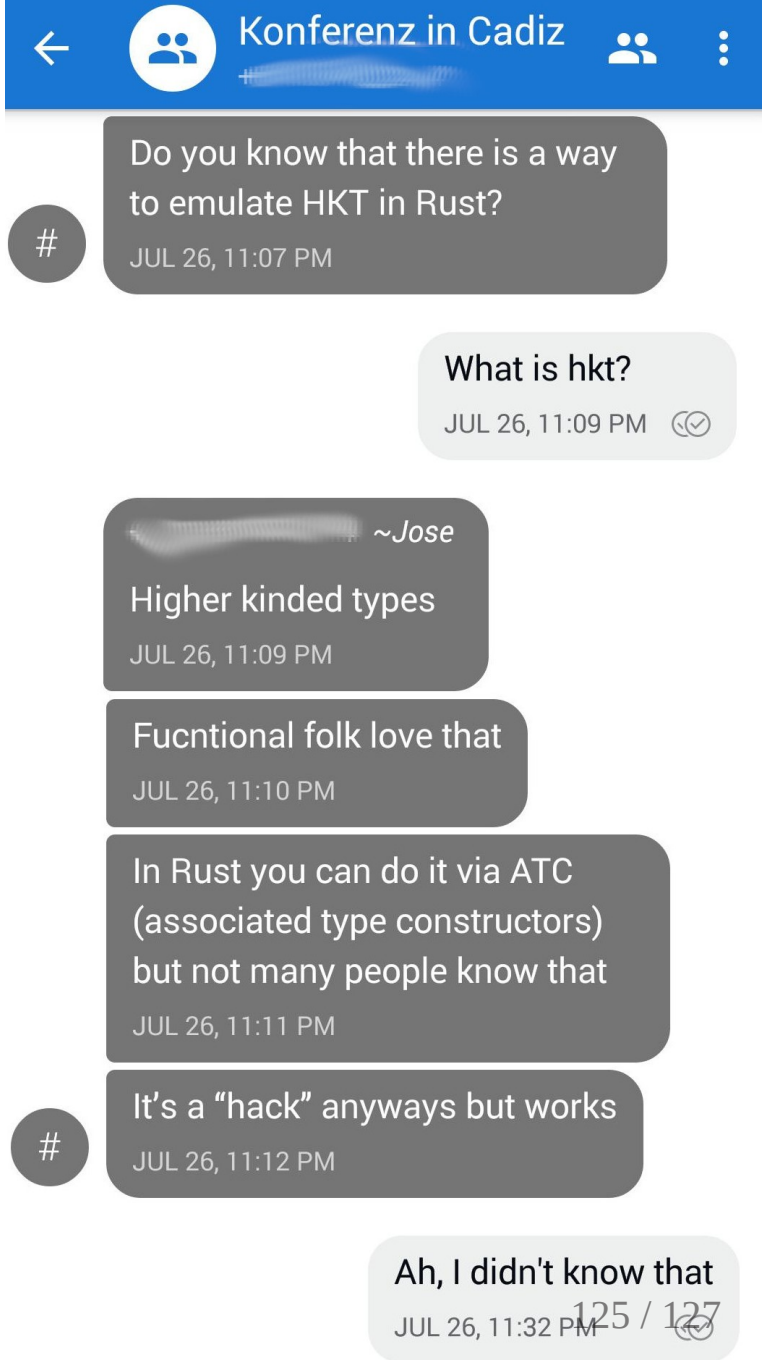
More nice things!

- Rust REPL RFC

More nice things!

- Rust REPL RFC
- Generic associated types RFC

  (formerly known as
  associated type constructors)



Where are the trees?

I thought of a pun when writing the talk title, but decided the pun doesn't work when writing the talk

¯_(ツ)_/¯

Thank you!

📧 mail@lislis.de

🐘 lislis@toot.cat

 https://github.com/lislis

📽 https://lislis.de/talks/lambda-world-2018/

Resources

- The Rust Programming Language
- Rust by Example
- Learn you a Haskell