

A Code Centric Journey Into the Gleam Language



Who am I

 Computer Science graduate at Unibo

 Functional programming enthusiast

 Member of the Gleam core team

What is Glean?



Gleam is a *friendly* language for
building *type-safe* systems that
scale!

gleam.run

```
import gleam/io
import gleam/list

pub fn main() {
  list.each(["Lucy", "Copenhagen"], greet)
  // Hello, Lucy!
  // Hello, Copenhagen!
}

fn greet(name: String) → Nil {
  io.println("Hello, " <> name <> "!")
}
```

```
import gleam/io
import gleam/list
```

```
pub fn main() {
  list.each(["Lucy", "Copenhagen"], greet)
  // Hello, Lucy!
  // Hello, Copenhagen!
}
```

```
fn greet(name: String) → Nil {
  io.println("Hello, " <> name <> " !")
}
```

```
import gleam/io
import gleam/list

pub fn main() {
  list.each(["Lucy", "Copenhagen"], greet)
  // Hello, Lucy!
  // Hello, Copenhagen!
}

fn greet(name: String) → Nil {
  io.println("Hello, " <> name <> "!")
}
```

```
import gleam/io
import gleam/list

pub fn main() {
  list.each(["Lucy", "Copenhagen"], greet)
  // Hello, Lucy!
  // Hello, Copenhagen!
}

fn greet(name: String) → Nil {
  io.println("Hello, " <> name <> " !")
}
```

```
import gleam/io
import gleam/list
```

```
pub fn main() {
  ["Lucy", "Copenhagen"]
  |> list.each(greet)
}
```

```
fn greet(name: String) → Nil {
  io.println("Hello, " <> name <> " !")
}
```

Gleam is type safe

- ✨ Gleam has a strong static type system
- 🔮 The compiler can always perform full type checking
- 😌 No implicit conversions

```
pub fn main() {  
    greet("Lucy")  
}
```

```
fn greet(name: String) → Nil {  
    io.println("Hello, " <> name <> "!" )  
}
```

```
pub fn main() {  
    greet("Lucy")  
}
```

```
fn greet(name: String) → Nil {  
    io.println("Hello, " <> name <> "!")  
}
```

```
pub fn main() {  
    greet("Lucy")  
}
```

```
fn greet(name: String) → Nil {  
    io.println("Hello, " <> name <> "!" )  
}
```

```
pub fn main() {  
    greet(11)  
}
```

```
fn greet(name: String) → Nil {  
    io.println("Hello, " <> name <> "!" )  
}
```

error: Type mismatch

4 | /src/main.gleam:4:8
 | greet(11)
 ^^

Expected type:

String

Found type:

Int

```
pub fn main() {  
    greet(11)  
}
```

There's **no hidden magic** that will turn an Int into a String



```
fn greet(name: String) → Nil {  
    io.println("Hello, " <> name <> "!!")  
}
```

```
pub fn main() {  
    greet(int.to_string(11))  
}
```

```
fn greet(name: String) → Nil {  
    io.println("Hello, " <> name <> "!!")  
}
```

Gleam is type safe

- ✨ Gleam has a strong static type system
- 🪄 The compiler can always perform full type checking
- 😌 No implicit conversions

Gleam is type safe

- ✨ Gleam has a strong static type system
- 🔮 The compiler can always perform full type checking
- 😌 No implicit conversions, **no null and no exceptions**

```
pub fn main() {  
    let response =  
        request.new()  
        |> request.set_host("pokeapi.co")  
        |> request.set_path("api/v2/pokemon/cleffa")  
        |> httpc.send  
  
    io.println(response.body)  
}
```

error: Unknown record field

/src/goto.gleam:12:20

12 io.debug(response.**body**)
 ^^^^^ This field does not exist

The value being accessed has this type:

Result(Response(String), Dynamic)

It does not have any fields.

error: Unknown record field

└─ /src/goto.gleam:12:20

12 ┌ io.debug(response.body)

^^^^ This field does not exist

The value being accessed has this type:

Result(Response(String), Dynamic)

It does not have any fields.

If something can fail it
has to return a
Result



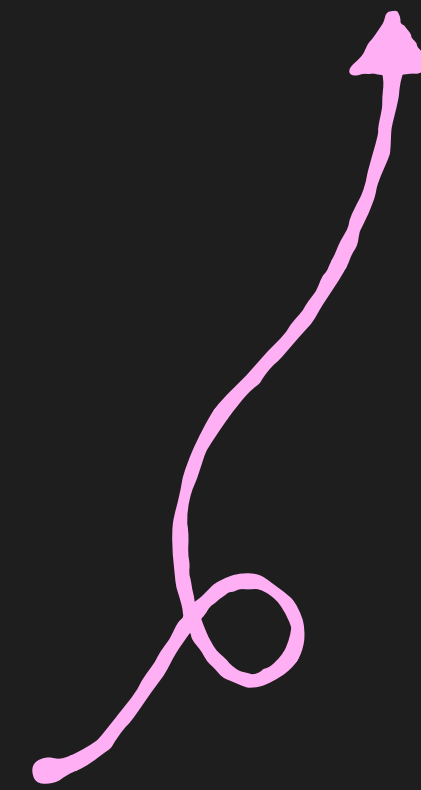
What's in a Result?

```
pub fn main() {  
    let ok: Result<Int, String> = Ok(11)  
    let error: Result<Int, String> = Error("Bad request!")  
}
```

What's in a Result?

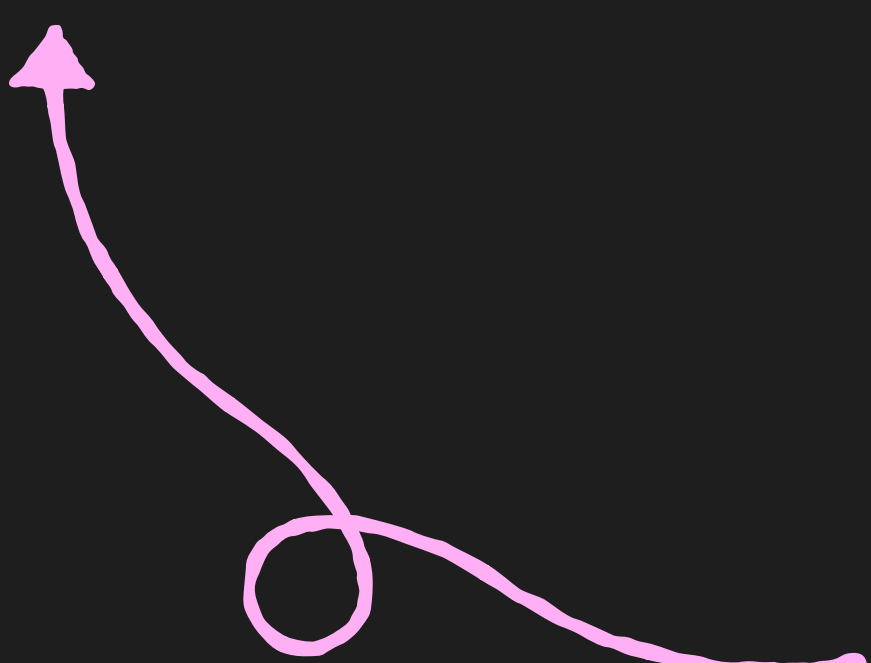
```
pub fn main() {  
    let ok: Result<Int, String> = Ok(11)  
    let error: Result<Int, String> = Error("Bad request!")  
}
```

There's **two ways to build a result**: **Ok**, to represent success and **Error** to represent failure



What's in a Result?

```
pub fn main() {  
    let ok: Result<Int, String> = Ok(11)  
    let error: Result<Int, String> = Error("Bad request!")  
}
```



Both `Ok` and `Error` produce a value of type **Result**, so when we get a `Result` back we need some way to know if it failed or not!

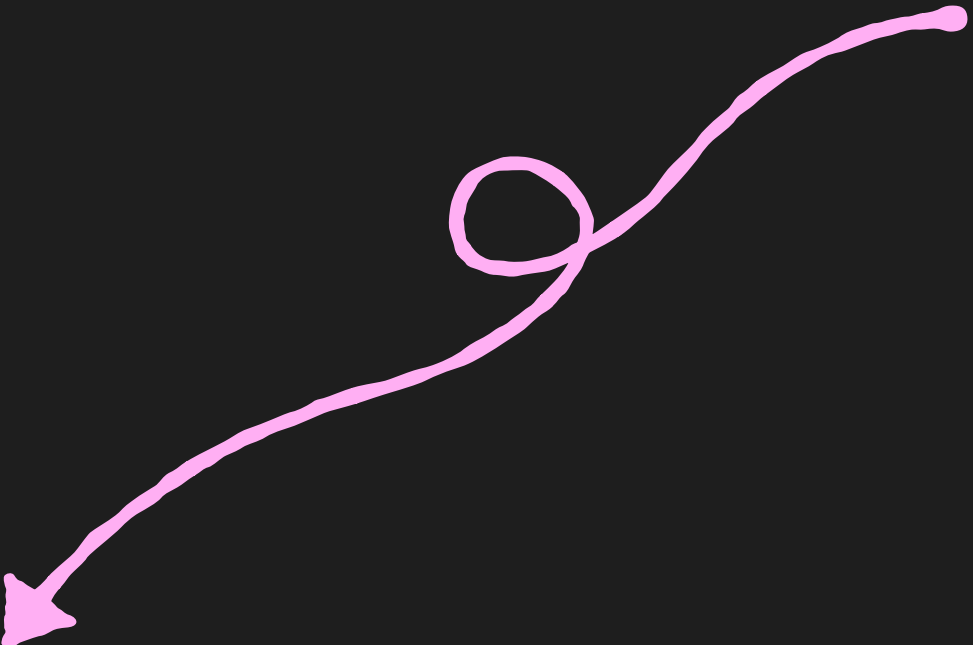
```
pub fn main() {  
    let response =  
        request.new()  
        |> request.set_host("pokeapi.co")  
        |> request.set_path("api/v2/pokemon/cleffa")  
        |> httpc.send  
  
    io.println(response.body)  
}
```

```
pub fn main() {  
    let request_result =  
        request.new()  
        |> request.set_host("pokeapi.co")  
        |> request.set_path("api/v2/pokemon/cleffa")  
        |> httpc.send  
  
    case request_result {  
        Ok(response) → io.println(response.body)  
        Error(_reason) → io.println("💣 An error occurred!")  
    }  
}
```

```
pub fn main() {  
    let request_result =  
        request.new()  
        |> request.set_host("pokeapi.co")  
        |> request.set_path("api/v2/pokemon/cleffa")  
        |> httpc.send  
  
    case request_result {  
        Ok(response) → io.println(response.body)  
        Error(_reason) → io.println("💣 An error occurred!")  
    }  
}
```

```
pub fn main() {  
    let request_result =  
        request.new()  
        |> request.set_host("pokeapi.co")  
        |> request.set_path("/")  
        |> httpc.send  
  
    case request_result {  
        Ok(response) → io.println(response.body)  
        Error(_reason) → io.println("💣 An error occurred!")  
    }  
}
```

If everything worked out fine we can finally access the server's response, **wrapped in an Ok**



```
pub fn main() {  
    let request_result =  
        request.new()  
        |> request.set_host("pokeapi.co")  
        |> request.set_path("api/v2/pokemon/cleffa")  
        |> httpc.send  
  
    case request_result {  
        Ok(response) → io.println(response.body)  
        Error(_reason) → io.println("💣 An error occurred!")  
    }  
}
```

We also have to **deal with any error** that might have occurred

```
case request_result {  
  Ok(response) → io.println(response.body)  
  // Uh oh ...  
}
```

error: Inexhaustive patterns

```
src/goto.gleam:15:3
15  case response {
16    Ok(response) → io.println(response.body)
17  }
    ^
```

This case expression does not have a pattern for all possible values.
If it is run on one of the values without a pattern then it will crash.

The missing patterns are:

`Error(_)`

1 + "hello"

~~~~~

Most of my other code feels like a liability on some level [...] like it's strewn with hidden landmines and booby traps. **Looking at my Gleam code makes me feel calm.**

*Eileen Noonan - "Lustre and Gleam Make my Heart Rate Go Down - a Case Study"*

# Gleam can scale

⚙️ Gleam compiles to Erlang and runs on the BEAM

💪 The BEAM is an amazing, battle-tested runtime

🌐 Designed for massive concurrency and distribution

```
fn spawn_task(i) {  
    task.async(fn() {  
        greet(int.to_string(i))  
    })  
}
```

**Processes are lightweight**, have a small memory footprint, are fast to create and terminate and have a low scheduling overhead

```
pub fn main() {  
    list.range(0, 200_000)  
    |> list.map(spawn_task)  
    |> list.each(task.await_forever)  
}
```



# Gleam can scale

- ✨ Gleam shines as part of a wider, richer ecosystem
- 🤝 FFI is incredibly straightforward

```
pub fn main() {  
    tensor_from_list([1, 2, 3])  
    |> add(tensor_from_list([4, 5, 6]))  
}
```

```
type Tensor
```

```
@external(erlang, "Elixir.Nx", "tensor")  
pub fn tensor_from_list(data: List(Int)) → Tensor
```

```
@external(erlang, "Elixir.Nx", "add")  
pub fn add(one: Tensor, other: Tensor) → Tensor
```

```
pub fn main() {  
  tensor_from_list([1, 2, 3])  
  ▷ add(tensor_from_list([4, 5, 6]))  
}
```

```
type Tensor
```

```
@external(erlang, "Elixir.Nx", "tensor")  
pub fn tensor_from_list(data: List(Int)) → Tensor
```

```
@external(erlang, "Elixir.Nx", "add")  
pub fn add(one: Tensor, other: Tensor) → Tensor
```

# Gleam can scale

✨ Gleam shines as part of a wider, richer ecosystem

🤝 FFI is incredibly straightforward

# Gleam can scale

- ✨ Gleam shines as part of a wider, richer ecosystem
- 🤝 FFI is incredibly straightforward
- ↻ Gleam code can be used by other BEAM languages too!

```
% You can call Gleam from Erlang ...
```

```
an_erlang_function() →  
  hello_goto:greet(~"Lucy").
```

```
# ... and from Elixir too!
```

```
def an_elixir_function(), do:  
  :hello_goto.greet("Lucy")  
end
```

# Gleam can scale

- ✨ Gleam shines as part of a wider, richer ecosystem
- 🤝 FFI is incredibly straightforward
- ↻ Gleam code can be used by other BEAM languages too!

# Gleam can scale

- ✨ Gleam shines as part of a wider, richer ecosystem
- 🤝 FFI is incredibly straightforward
- ↻ Gleam code can be used by other BEAM languages too!
- 💣 Gleam compiles to JavaScript as well

# Gleam is friendly

⚠ The compiler produces clear and helpful error messages

error: Unknown record field

```
src/goto.gleam:15:3
15  user.nam
      ^^^^ Did you mean `name`?
```

The value being accessed has this type:

User

It has these fields:

.name

.id

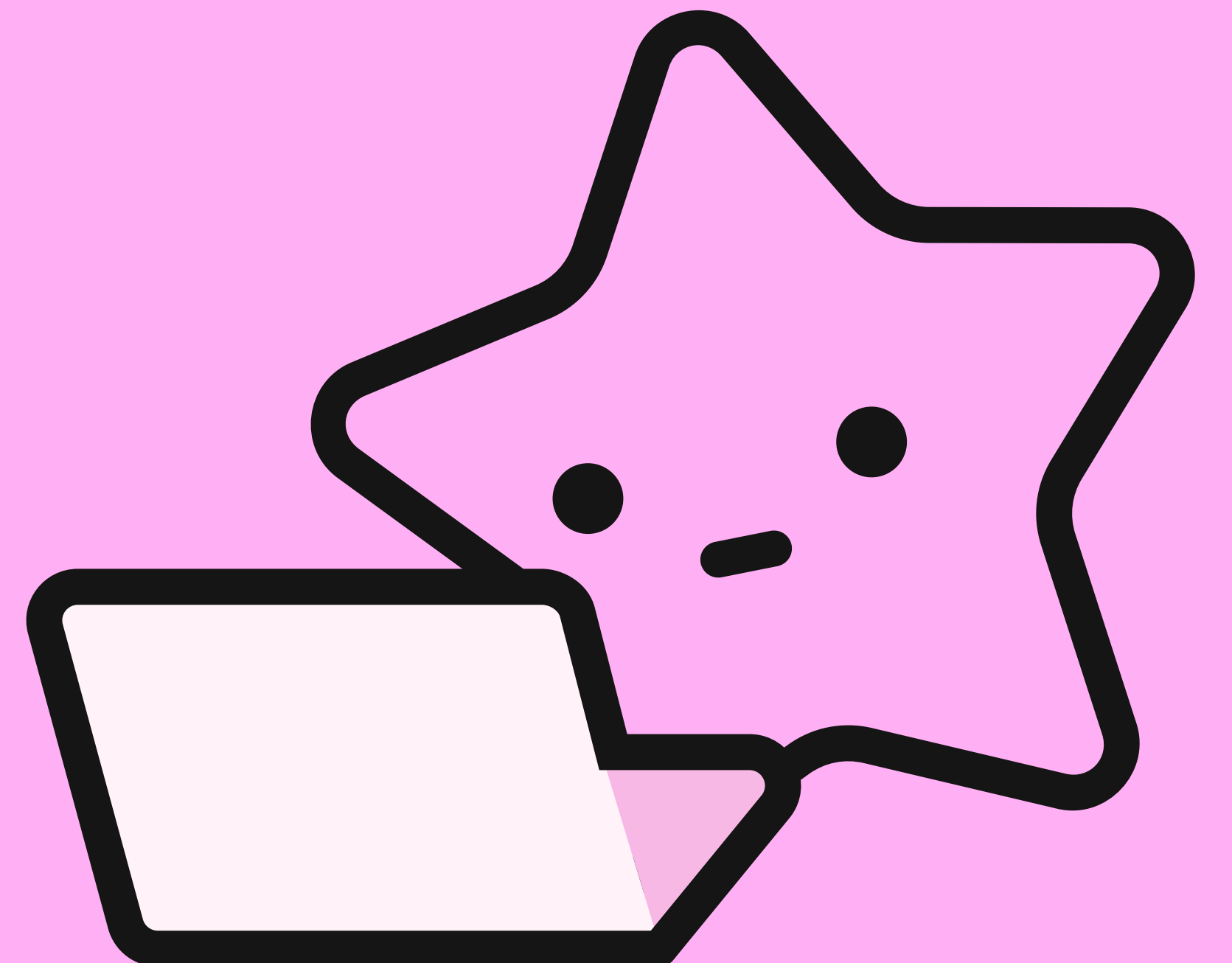
# Gleam is friendly

⚠ The compiler produces clear and helpful error messages

# Gleam is friendly

- ⚠️ The compiler produces clear and helpful error messages
- 📦 Gleam comes with compiler, build tool and package manager built in
- 🚒 Gleam comes with a built in formatter that has zero configuration
- 🔧 Gleam comes with a built in language server

# Show, don't tell!

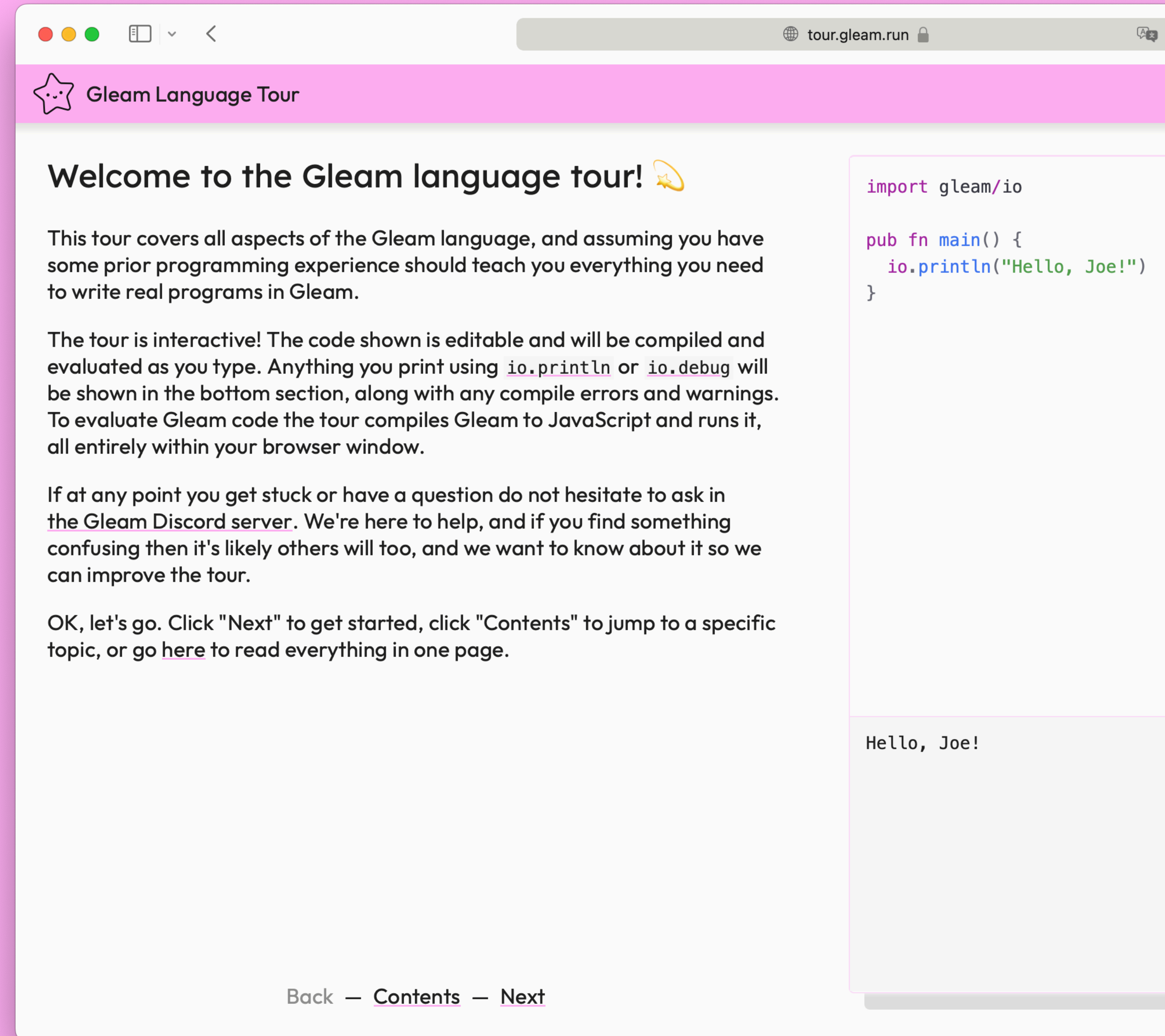


# What now?



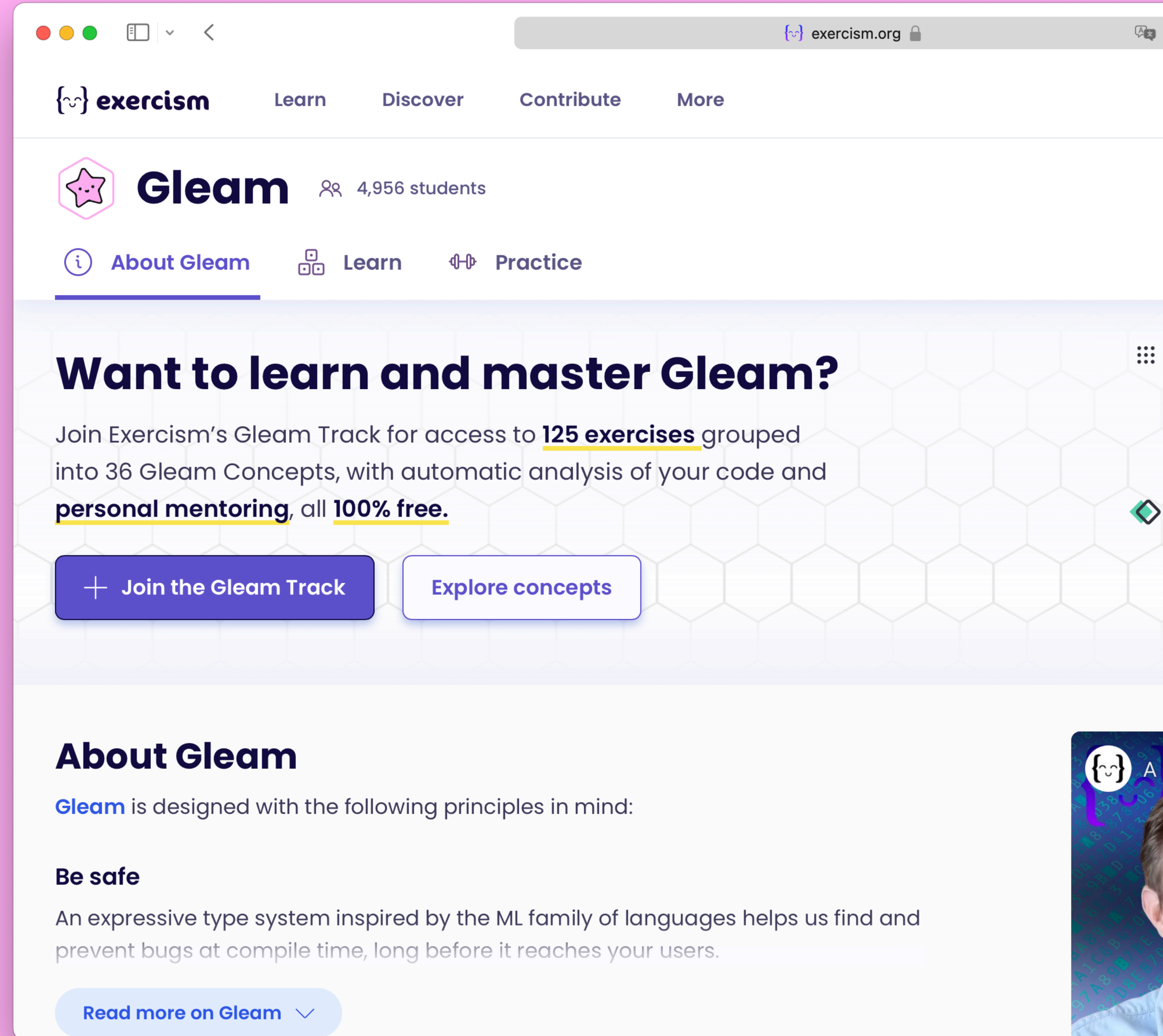
# Gleam language tour

tour.gleam.run



# Gleam Exercism track

[exercism.org/tracks/gleam](https://exercism.org/tracks/gleam)



The screenshot shows the Exercism website's Gleam track page. At the top, the Exercism logo and navigation links (Learn, Discover, Contribute, More) are visible. Below this, the Gleam track is highlighted with a star icon and the text '4,956 students'. A sub-navigation bar includes 'About Gleam' (selected), 'Learn', and 'Practice'. The main content area features a large heading 'Want to learn and master Gleam?' followed by a paragraph: 'Join Exercism's Gleam Track for access to 125 exercises grouped into 36 Gleam Concepts, with automatic analysis of your code and personal mentoring, all 100% free.' Two buttons are present: 'Join the Gleam Track' and 'Explore concepts'. Below this, the 'About Gleam' section begins with the text 'Gleam is designed with the following principles in mind:' and a sub-section 'Be safe' which states: 'An expressive type system inspired by the ML family of languages helps us find and prevent bugs at compile time, long before it reaches your users.' A 'Read more on Gleam' button with a dropdown arrow is at the bottom. A small sidebar on the right shows a user profile icon and the letter 'A'.

exercism

Learn Discover Contribute More

 **Gleam** 4,956 students

 About Gleam  Learn  Practice

## Want to learn and master Gleam?

Join Exercism's Gleam Track for access to 125 exercises grouped into 36 Gleam Concepts, with automatic analysis of your code and personal mentoring, all 100% free.

[+ Join the Gleam Track](#) [Explore concepts](#)

### About Gleam

**Gleam** is designed with the following principles in mind:

#### Be safe

An expressive type system inspired by the ML family of languages helps us find and prevent bugs at compile time, long before it reaches your users.

[Read more on Gleam](#) ▾

# Get in touch!

Join Glean's community on Discord

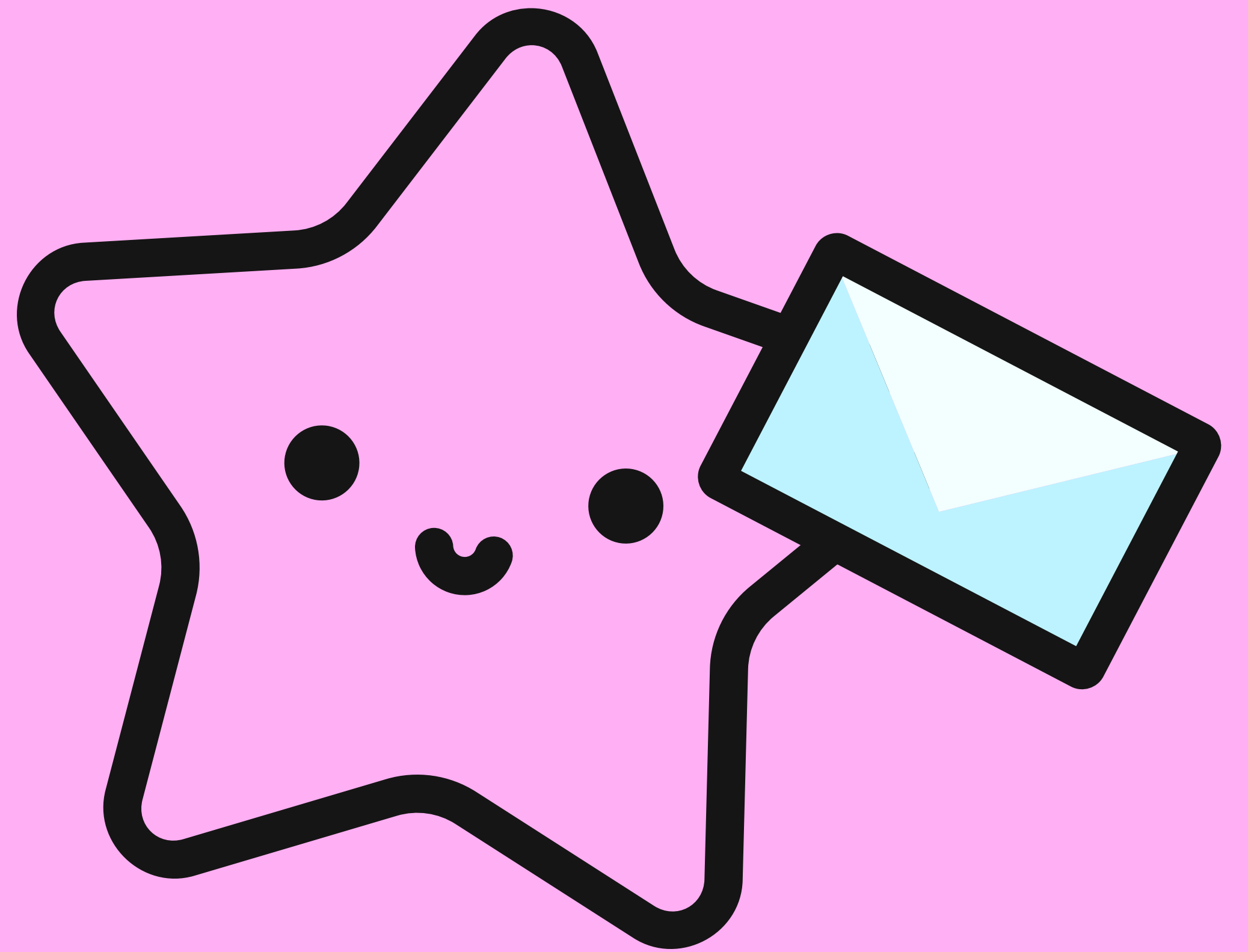


# Ping me!

✉ [giacomo.cavalieri@icloud.com](mailto:giacomo.cavalieri@icloud.com)

🐦 [@giacomo\\_cava](https://twitter.com/@giacomo_cava)

🐙 [giacomocavalieri](https://github.com/giacomocavalieri)



# Thanks for listening!

Please rate this session in the Goto app

