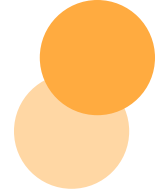


Enjoy Typesafe Web Development with Eta

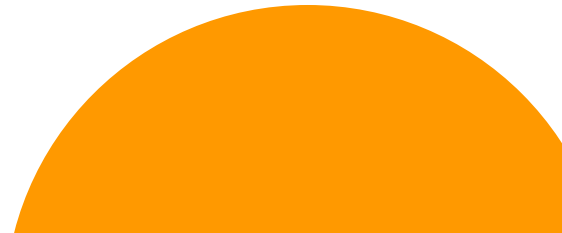
Abdurrachman Mappuji
Kukul Tech
Conf42 - Java2021





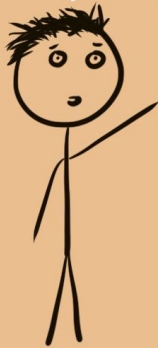
Outline

- ❑ What is **functional programming**?
- ❑ What is the **benefit** of functional programming?
- ❑ **Types** of functional programming **languages**
- ❑ What functional programming **success** I ever encountered? **Elm**
- ❑ What I want to explore? Pure functional programming in **JVM**
- ❑ **Quick Demo** of Eta, Pure FP language in JVM



Frontend development conferences

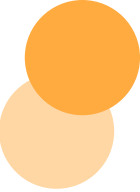
Look at this amazing concept created last night!

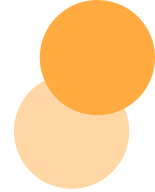


Functional programming conferences

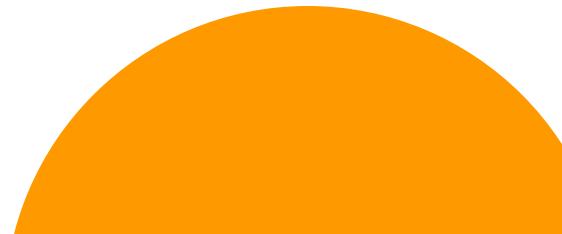
Look at this amazing concept created in 1978!







What is **functional programming**?



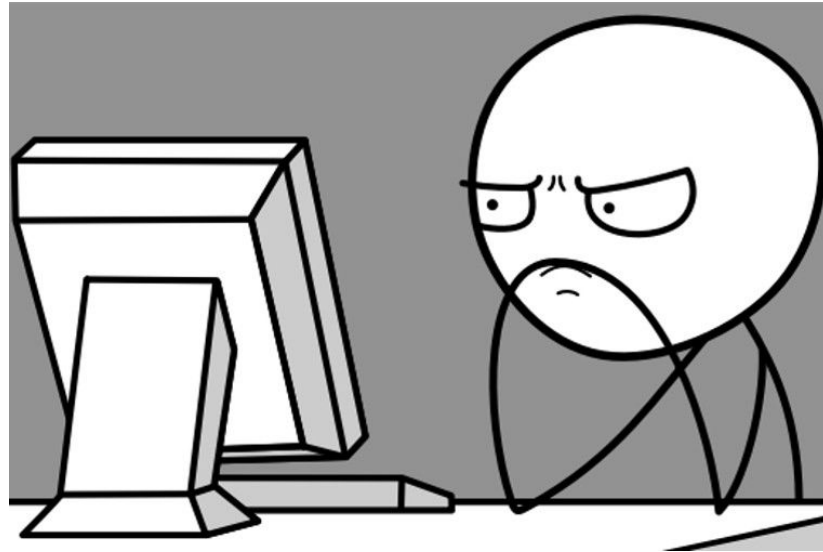
This is John, he is a software engineer



John is writing a webapp, so he grab his favorite language and start writing



He's using a new tech stack, so it will take a while...





**A FEW
MOMENTS LATER**

Cool, now its working and he's ready to create a pull request



Now, the code will go through a series of code review



He got a recommendation to refactor the code



I Am Developer

@iamdeveloper

Follow



10 lines of code = 10 issues.

500 lines of code = "looks fine."

Code reviews.

1:58 AM - 5 Nov 2013

8,035 Retweets 4,401 Likes



106



8.0K



4.4K

Cool, he now refactored the code. But wait, it breaks the app 🤔

PRIORITIES



What we learnt from here?

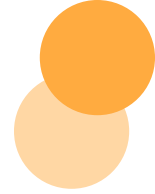
- ❑ Developing software is hard
- ❑ Building a reliable software needs a lot of effort e.g. code reviews
- ❑ Refactor is good, but can it be more simple?
- ❑ Is it possible to make refactoring experience more enjoyable?

🏆 FUNCTIONAL
PROGRAMMING TO THE
RESCUE 🏆

Welcome to Functional Programming

How 1978 concept can help you code better?

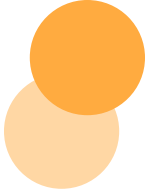
- ❑ Make your mind to
 - ❑ See things differently
 - ❑ Be more creative with composing functions (code)
- ❑ Write better code by
 - ❑ Make your code easy to read and understand (readability)
 - ❑ Make your code easy to change (refactoring)
 - ❑ Make your code scales better
 - ❑ Reduce problems when you scale your apps e.g. concurrency, deadlock, etc.



“All race conditions, deadlock conditions, and concurrent update problems are due to mutable variables.”

— Robert C. Martin, Clean Architecture





“Functional programming (often abbreviated FP) is the process of building software by composing pure functions, avoiding shared state, mutable data, and side-effects. Functional programming is declarative rather than imperative, and application state flows through pure functions.”

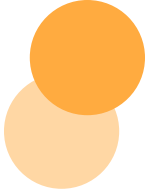
—Eric Elliott





What is the benefit of functional programming?



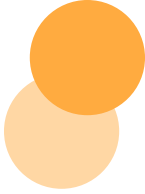


So what are the benefits?

Functional programming offers the following advantages –

- ❑ “Bugs-Free” Code – No side effect.
- ❑ Efficient Parallel Programming – Immutable.
- ❑ Efficiency – can take advantage of concurrent programming.
- ❑ Supports Nested Functions – Functional programming supports Nested Functions.
- ❑ Lazy Evaluation – Functional programming supports Lazy Functional Constructs like Lazy Lists, Lazy Maps, etc.



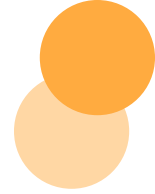


Types of functional programming languages



TL;DR

- ❑ Pure Functional Programming Language
 - ❑ Haskell
 - ❑ Elm
 - ❑ Mercury
 - ❑ Clean
 - ❑ etc
- ❑ Impure Functional Programming Language
 - ❑ Scala
 - ❑ Clojure
 - ❑ F#
 - ❑ OCaml
 - ❑ etc

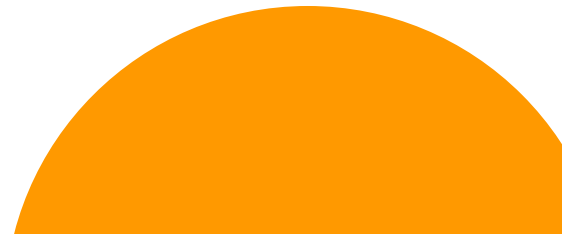
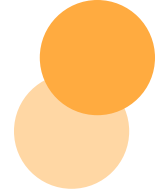


What functional programming
success I ever encountered? Elm



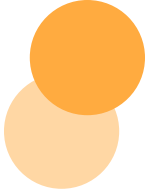
Elm - the aims

“A delightful language for reliable web applications.”



Elm - the testimonials

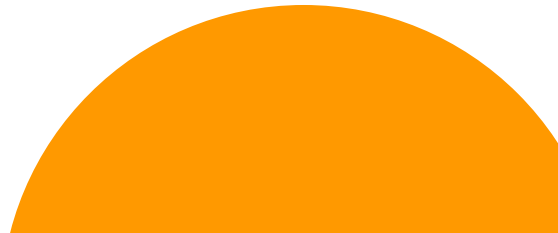
- ❑ *“It is the most productive programming language I have used.”*
—Rupert Smith, Software Engineer, The Sett Ltd
- ❑ *“[My favorite thing] is the feeling of joy and relaxation when writing Elm code.”*
—Luca Mugnaini, Software Engineer, Rakuten
- ❑ *“Using Elm, I can deploy and go to sleep!”*
—Mario Uher, CTO, yodel.io
- ❑ *“You just follow the compiler errors and come out the other end with a working app.”*
James Carlson, Software engineer

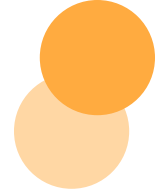


No Runtime Exceptions

Elm uses type inference to detect corner cases and give friendly hints. NoRedInk switched to Elm about four years ago, and 300k+ lines later, they still have not had to scramble to fix a confusing runtime exception in production. [Learn more.](#)


```
-- TYPE MISMATCH ----- Main.elm
The 1st argument to `drop` is not what I expect:
8| List.drop (String.toInt userInput) [1,2,3,4,5,6]
This `toInt` call produces:
    Maybe Int
But `drop` needs the 1st argument to be:
    Int
Hint: Use Maybe.withDefault to handle possible errors.
```





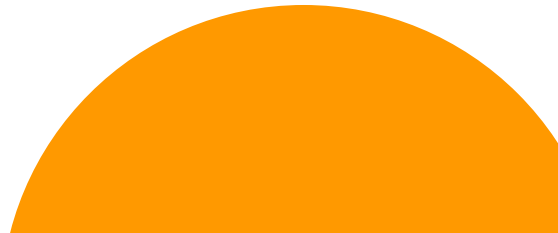
Fearless refactoring

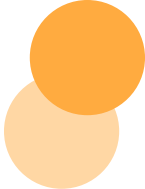
The compiler guides you safely through your changes, ensuring confidence even through the most wide-reaching refactorings in unfamiliar codebases.

Two light blue speech bubble icons, one slightly overlapping the other, positioned above the quote text.

Whether it's renaming a function or a type, or making a drastic change in a core data type, you just follow the compiler errors and come out the other end with a working app.

James Carlson, Software engineer

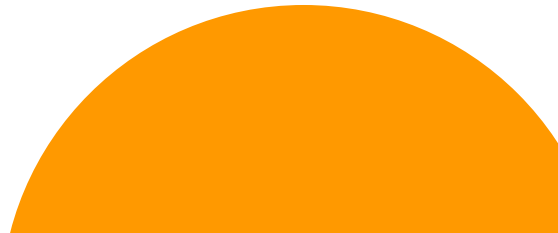


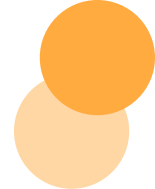


Understand anyone's code

Including your own, six months later. All Elm programs are written in the same pattern, eliminating doubt and lengthy discussions when deciding how to build new projects and making it easy to navigate old or foreign codebases. [Learn more.](#)

```
-- THE ELM ARCHITECTURE
init : ( Model, Cmd Msg )
update : Msg -> Model -> ( Model, Cmd Msg )
subscriptions : Model -> Sub Msg
view : Model -> Html Msg
```





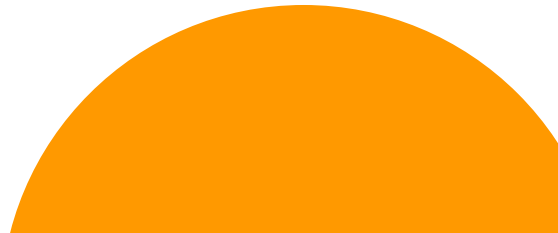
Fast and friendly feedback

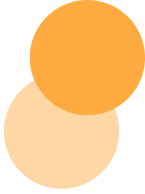
Enjoy Elm's [famously helpful](#) error messages. Even on codebases with hundreds of thousands lines of code, compilation is done in a blink.



I love how fast Elm is. I make a change and I get an immediate response. It's like I'm having a conversation with the compiler about how best to build things.

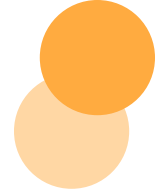
Wolfgang Schuster, Software engineer



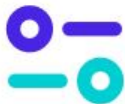


What I want to explore? Eta, Pure
functional programming in JVM





Why Eta?



Robust Interoperability

Eta has a strongly-typed Foreign Function Interface (FFI) that allows you to safely interoperate with Java.



Ultra Type Safety

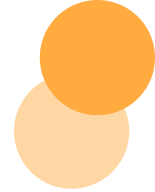
Eta has global type inference, giving you a dynamic language experience, but with a strong typing hidden underneath.



Scalable Concurrency Support

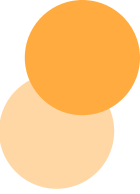
Eta offers a wide range of strategies for handling concurrency including Software Transaction Memory (STM), MVars, and Fibers.





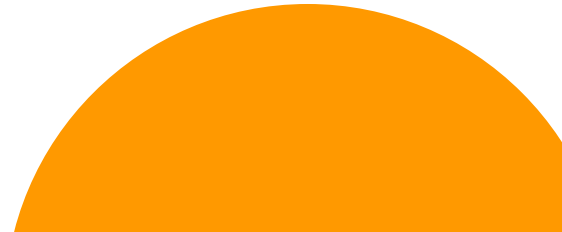
Quick Demo of Eta, Pure FP language in JVM





Let's see the Demo

<https://github.com/empeje/conf42-web-dev-eta>



The Eta Downside

- ❑ The language is not that maintainable anymore
- ❑ Managing compatibility with Haskell ecosystem is hard