finstack

# Play with Elm!

Rethink Web Development

## About Me

Choucri FAHED

CTO of Finstack

Scala, FP, Elm, Idris, Blockchains, AI, Universal Income... good food

## Summary

- Why Elm?
- What's Elm?
- TEA (The Elm Architecture)
- Elm vs ScalaJS
- Elm in Scala Projects

## Why Elm?

Created by Evan Czaplicki in 2012

*"If typed functional programming is so great, how come nobody uses it?"* Let's be mainstream!

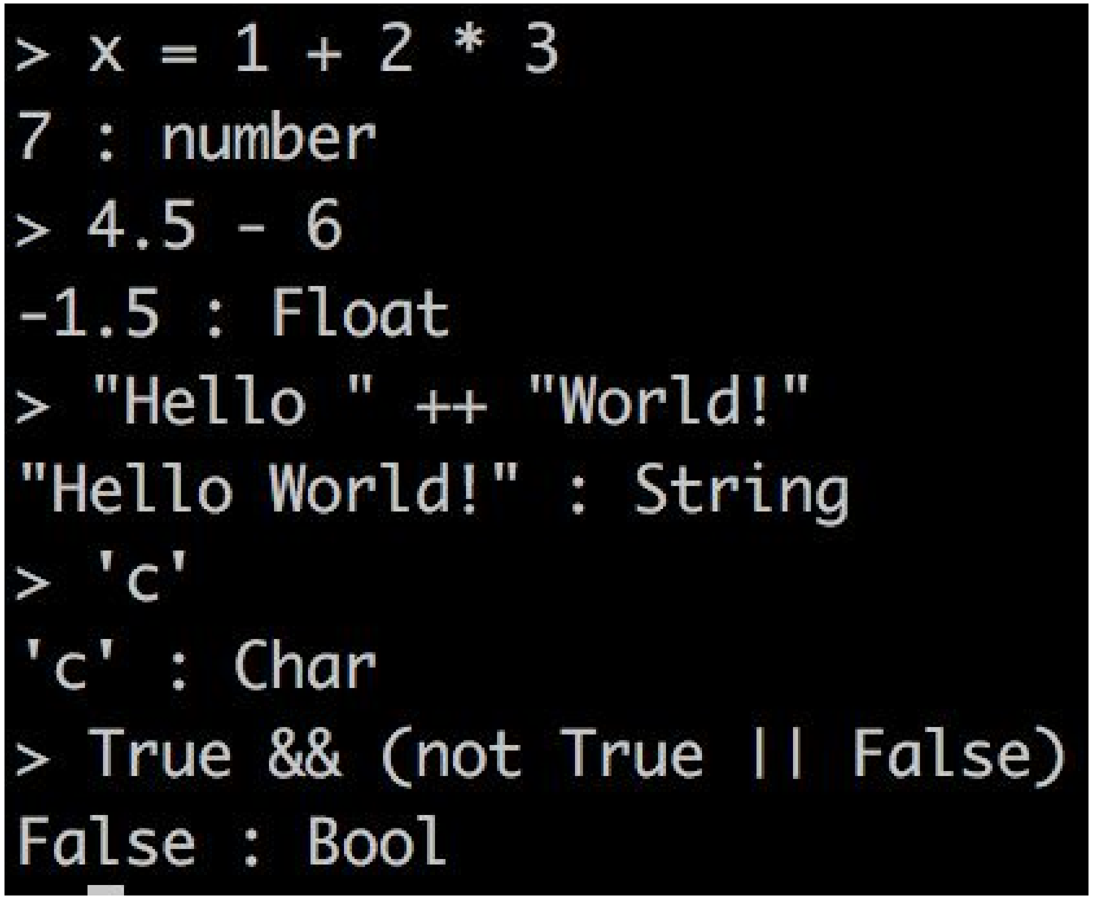
How it feels to learn JavaScript in 2016

Objective: NO RUNTIME ERRORS!

## What's Elm?

- **Simple** language
  - Statically typed, strict, purely functional
  - Compiles to JavaScript
- **Simple** framework / libraries
  - TEA (The Elm Architecture)
  - Use *"Obvious names"* No M-word
- **Simple** and user focused tools
  - Friendly compiler error messages
  - Time travel debugger, elm-format...
  - Automatic semver enforcement with elm-package

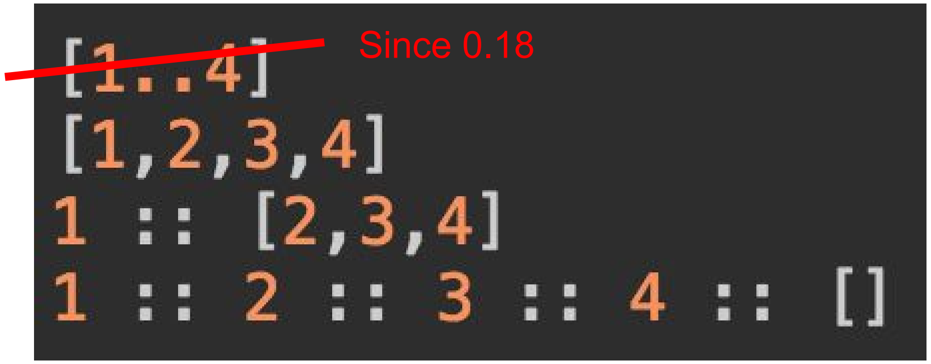
## Values - Literals



```
> x = 1 + 2 * 3
7 : number
> 4.5 - 6
-1.5 : Float
> "Hello " ++ "World!"
"Hello World!" : String
> 'c'
'c' : Char
> True && (not True || False)
False : Bool
```

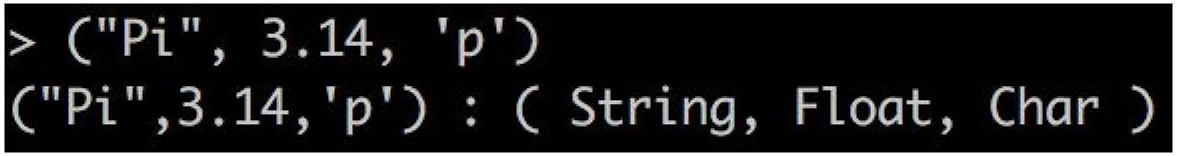
## Values - Lists & Tuples

- Lists: All 4 are equivalent below



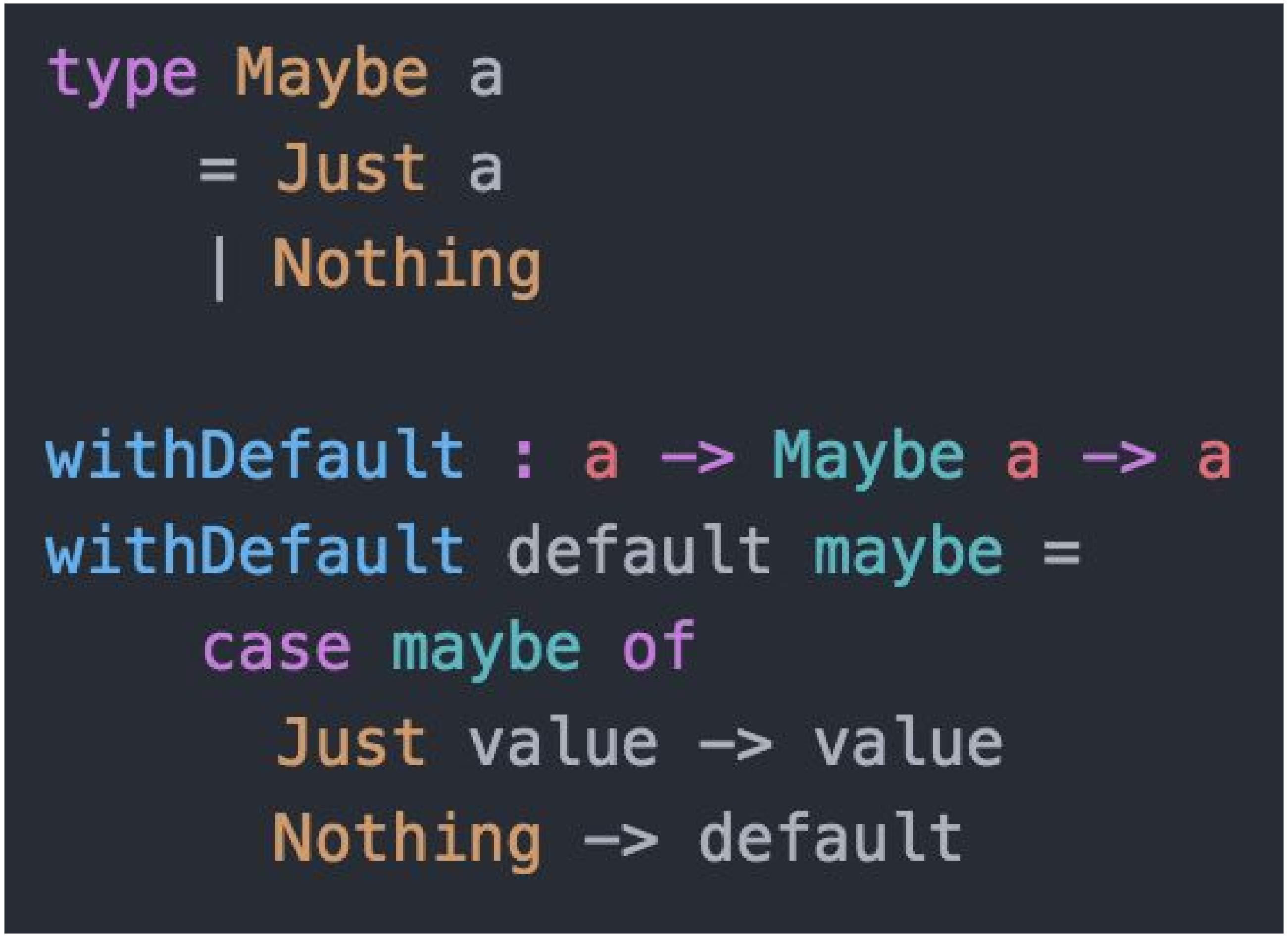
```
~~[1..4]~~  Since 0.18
[1,2,3,4]
1 :: [2,3,4]
1 :: 2 :: 3 :: 4 :: []
```

- Tuples



```
> ("Pi", 3.14, 'p')
("Pi",3.14,'p') : ( String, Float, Char )
```

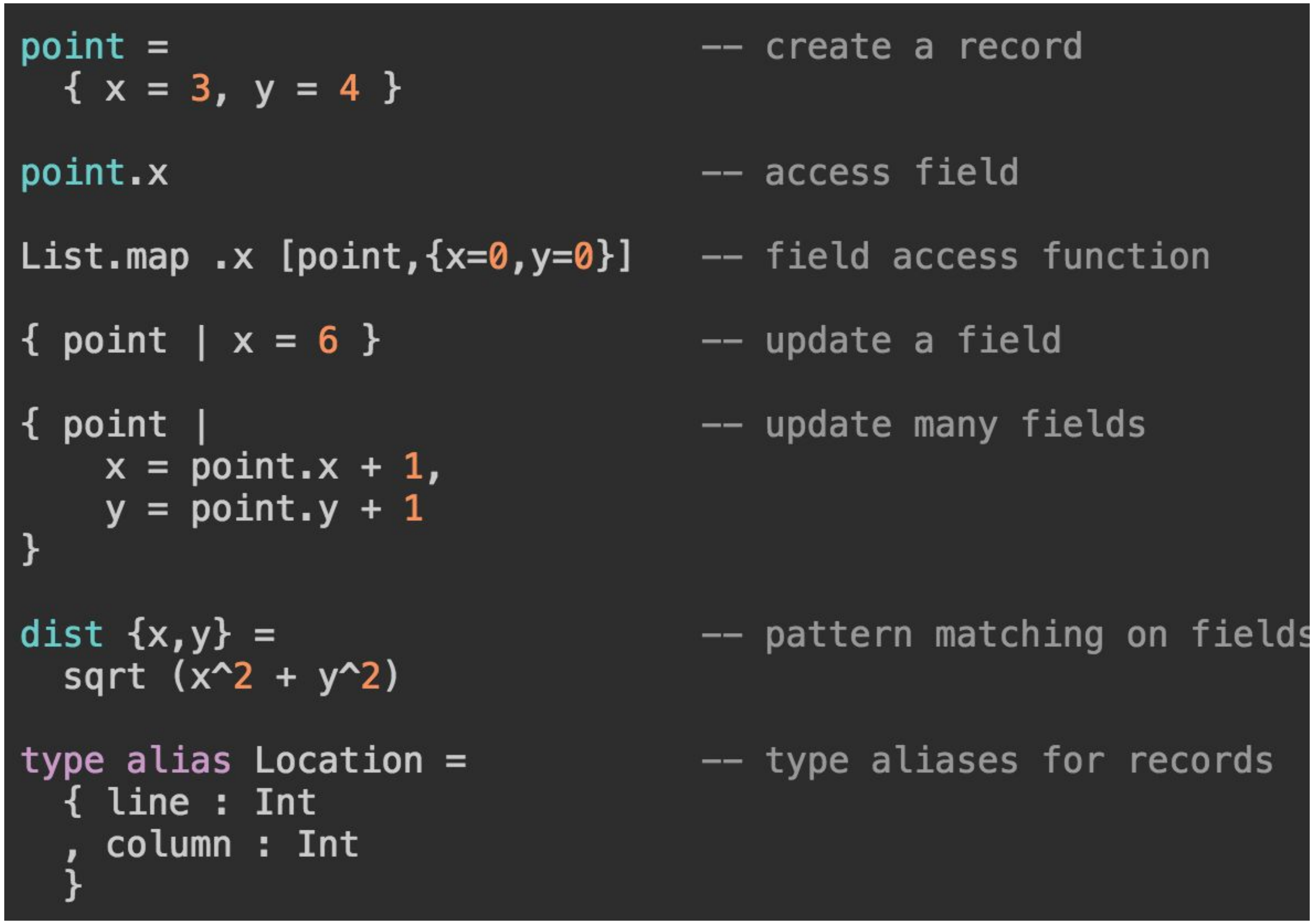
## Values - Union Types



```elm
type Maybe a
    = Just a
    | Nothing

withDefault : a -> Maybe a -> a
withDefault default maybe =
    case maybe of
      Just value -> value
      Nothing -> default
```

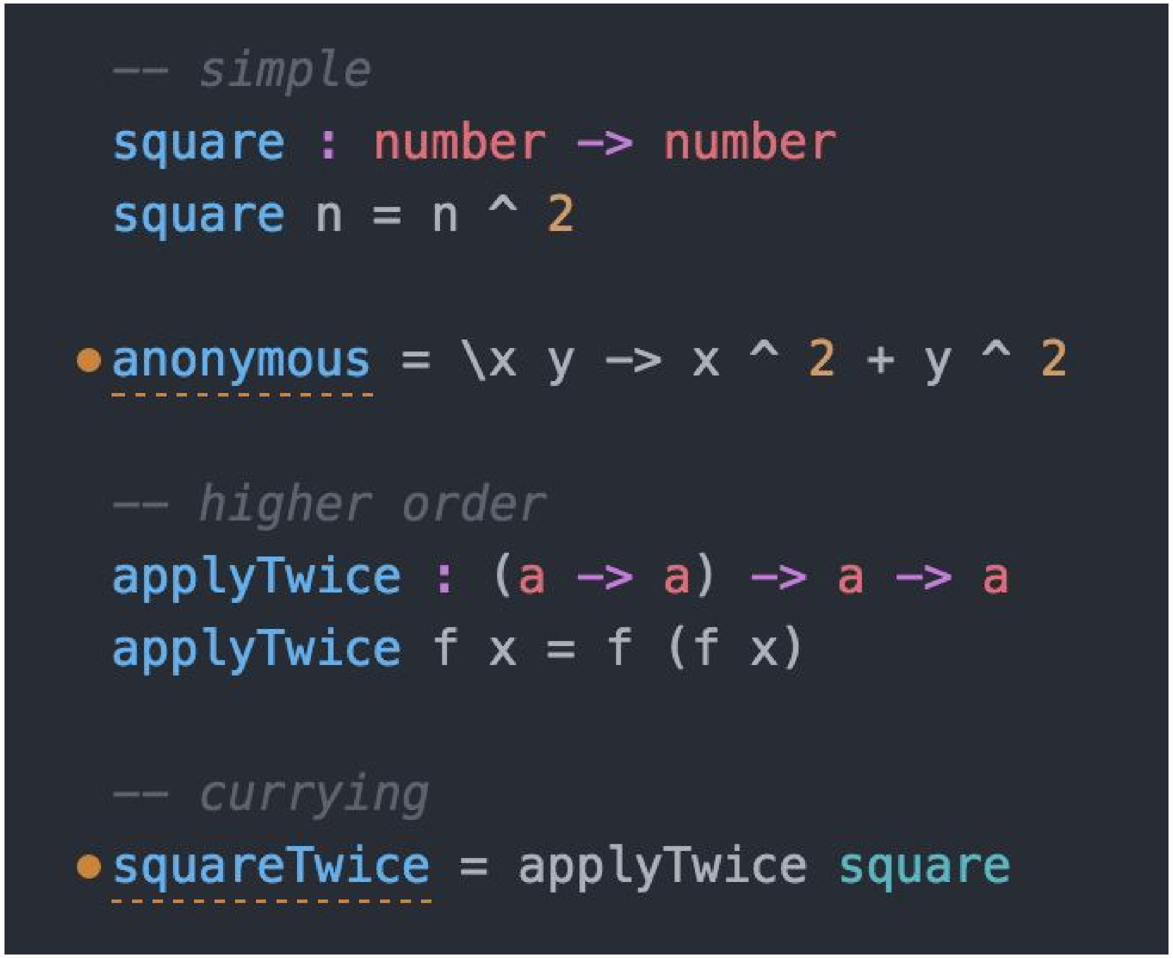
finstack

## Values - Records



```
point =                                 -- create a record
  { x = 3, y = 4 }

point.x                                 -- access field

List.map .x [point,{x=0,y=0}]           -- field access function

{ point | x = 6 }                       -- update a field

{ point |                               -- update many fields
    x = point.x + 1,
    y = point.y + 1
}

dist {x,y} =                            -- pattern matching on fields
  sqrt (x^2 + y^2)

type alias Location =                   -- type aliases for records
  { line : Int
  , column : Int
  }
```

# Functions



```
-- simple
square : number -> number
square n = n ^ 2

anonymous = \x y -> x ^ 2 + y ^ 2

-- higher order
applyTwice : (a -> a) -> a -> a
applyTwice f x = f (f x)

-- curring
squareTwice = applyTwice square
```

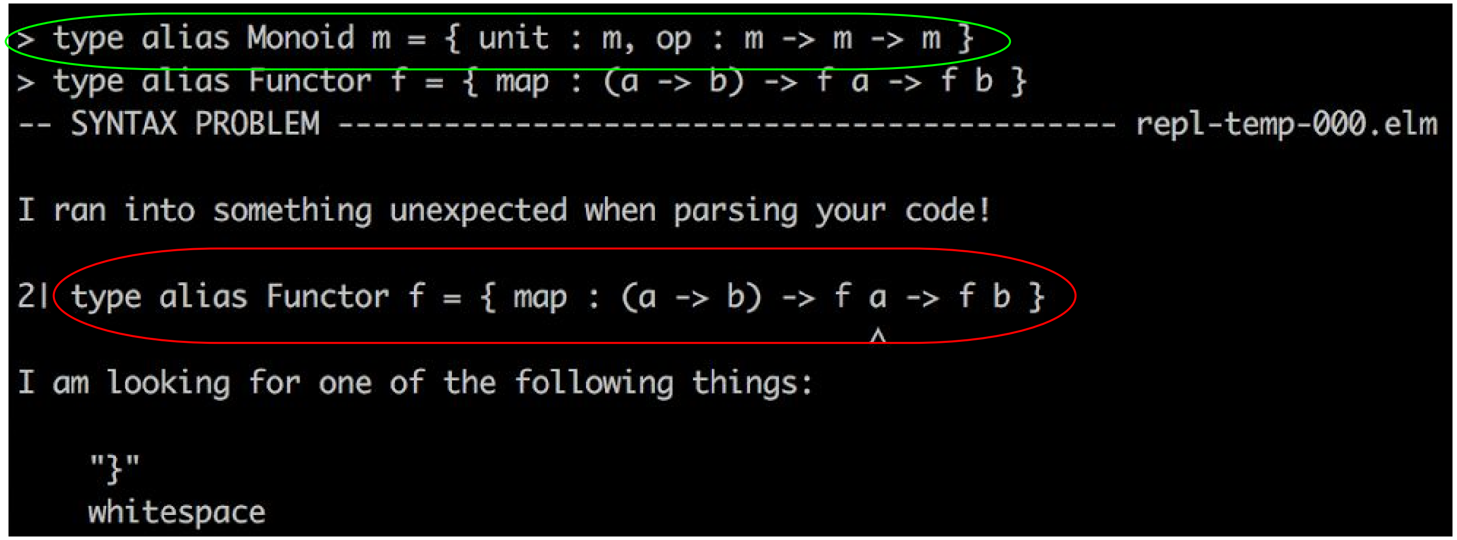
## What else?

- *If* and *let* expressions
- No *for* or *while* loops
- Operators
- JavaScript interop with *ports*
- Modules

For more Elm syntax

## That’s it?!

- No type classes
- No higher kinded types (ex: Functor[F[_]])
    - First order kinds like Monoids supported only



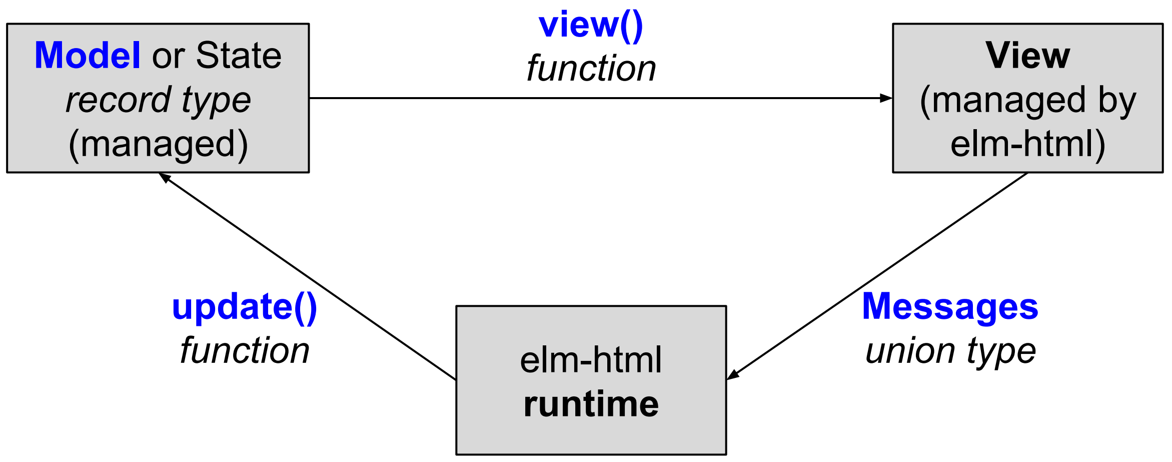
```
> type alias Monoid m = { unit : m, op : m -> m -> m }
> type alias Functor f = { map : (a -> b) -> f a -> f b }
-- SYNTAX PROBLEM ------------------------------------------- repl-temp-000.elm

I ran into something unexpected when parsing your code!

2| type alias Functor f = { map : (a -> b) -> f a -> f b }
                                                ^

I am looking for one of the following things:

    "}"
    whitespace
```

Feel frustrated? Try PureScript

# TEA - The Elm Architecture

## Simple and amazingly scalable pattern

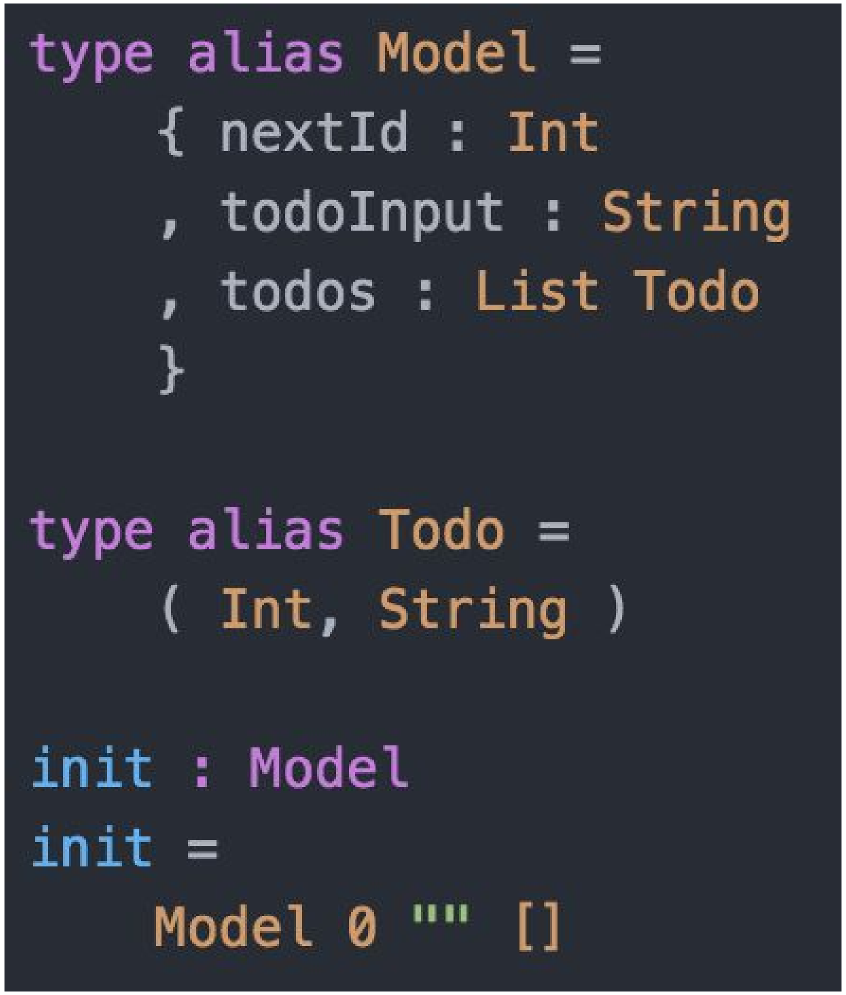
(overly simplified view below)



**code you need to write in blue*

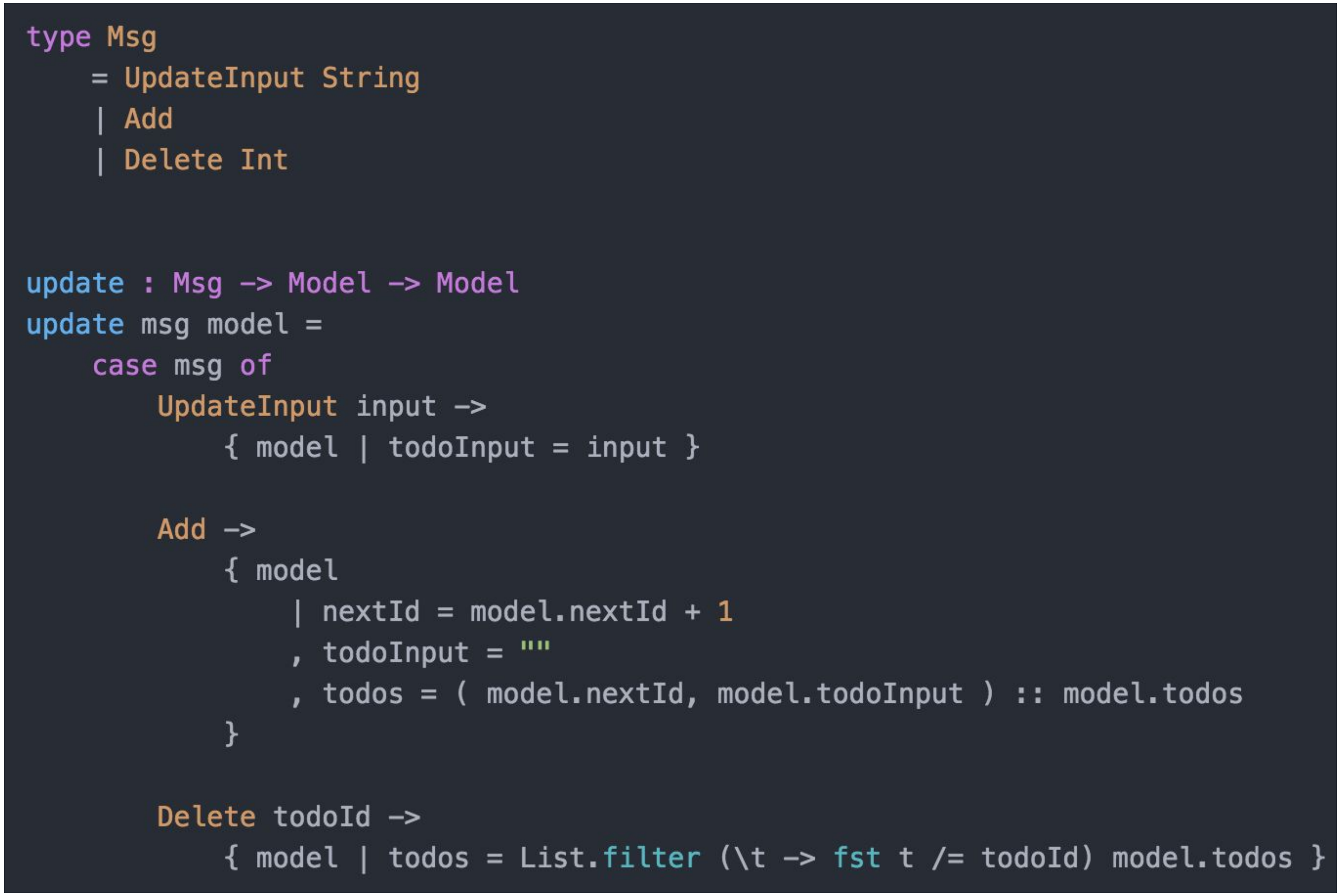
finstack

## TEA - Model


```elm
type alias Model =
    { nextId : Int
    , todoInput : String
    , todos : List Todo
    }


type alias Todo =
    ( Int, String )


init : Model
init =
    Model 0 "" []
```

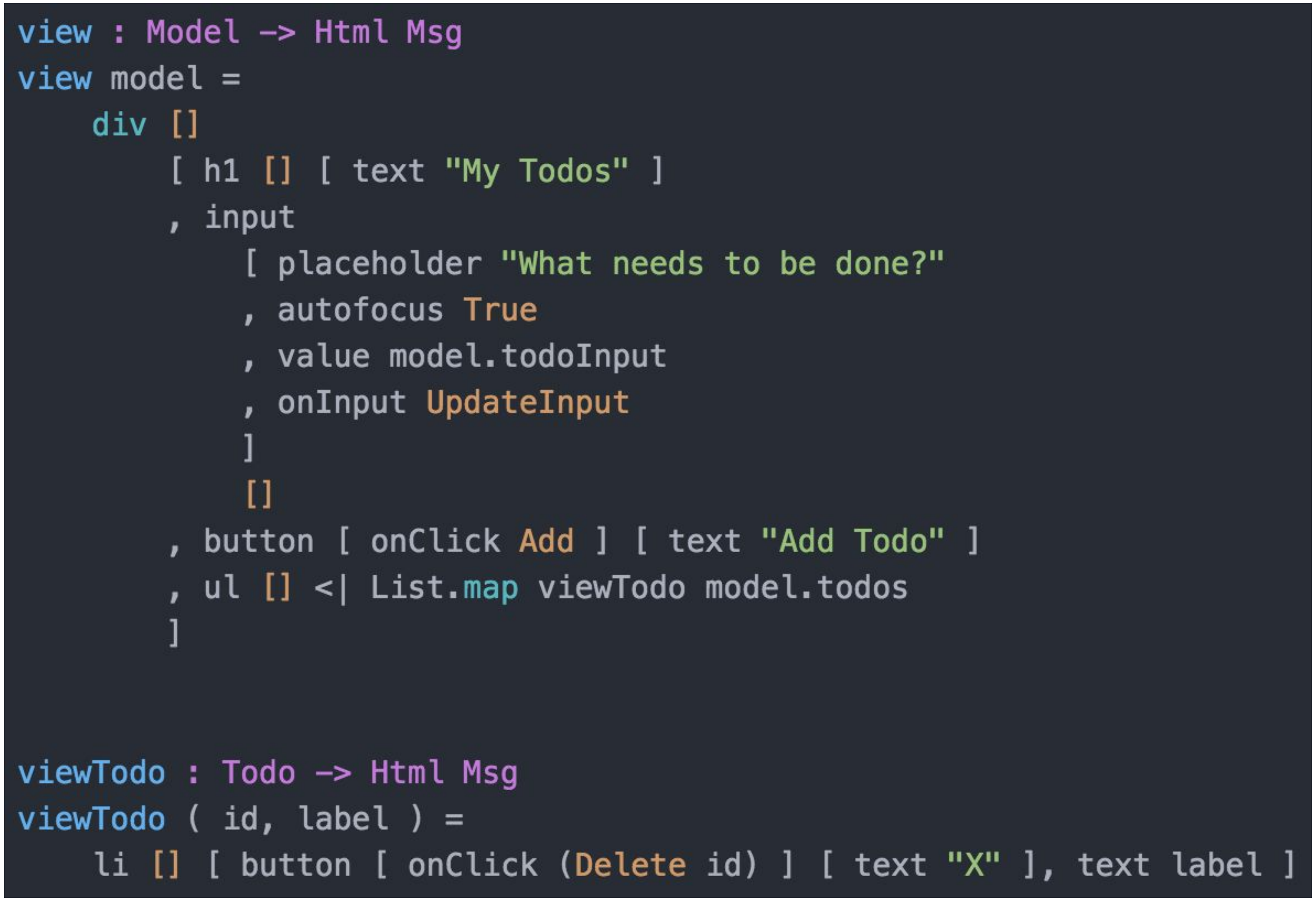
finstack

## TEA - Update


```elm
type Msg
    = UpdateInput String
    | Add
    | Delete Int


update : Msg -> Model -> Model
update msg model =
    case msg of
        UpdateInput input ->
            { model | todoInput = input }

        Add ->
            { model
                | nextId = model.nextId + 1
                , todoInput = ""
                , todos = ( model.nextId, model.todoInput ) :: model.todos
            }

        Delete todoId ->
            { model | todos = List.filter (\t -> fst t /= todoId) model.todos }
```

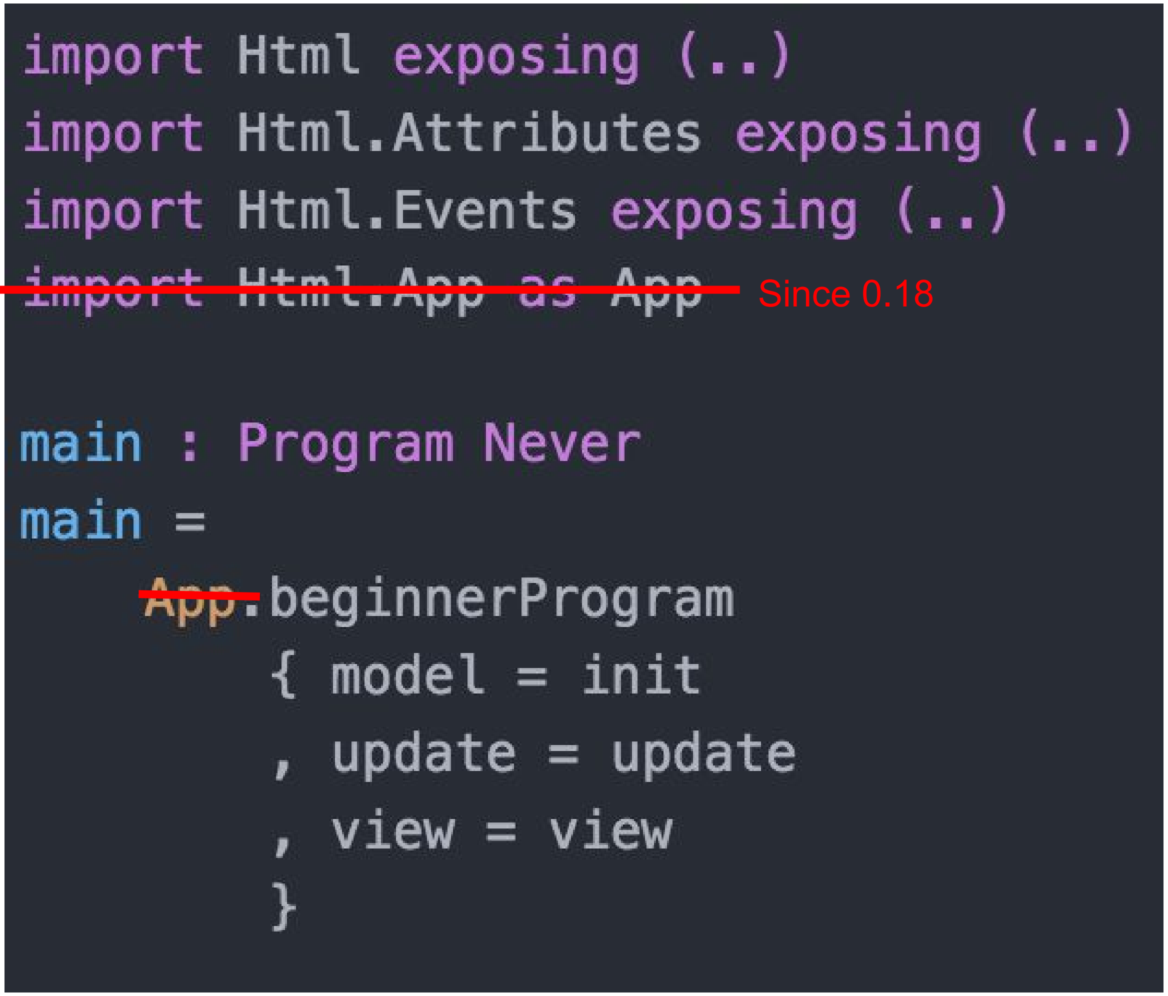
finstack

## TEA - View



```
view : Model -> Html Msg
view model =
    div []
        [ h1 [] [ text "My Todos" ]
        , input
            [ placeholder "What needs to be done?"
            , autofocus True
            , value model.todoInput
            , onInput UpdateInput
            ]
            []
        , button [ onClick Add ] [ text "Add Todo" ]
        , ul [] <| List.map viewTodo model.todos
        ]


viewTodo : Todo -> Html Msg
viewTodo ( id, label ) =
    li [] [ button [ onClick (Delete id) ] [ text "X" ], text label ]
```

# TEA - Main



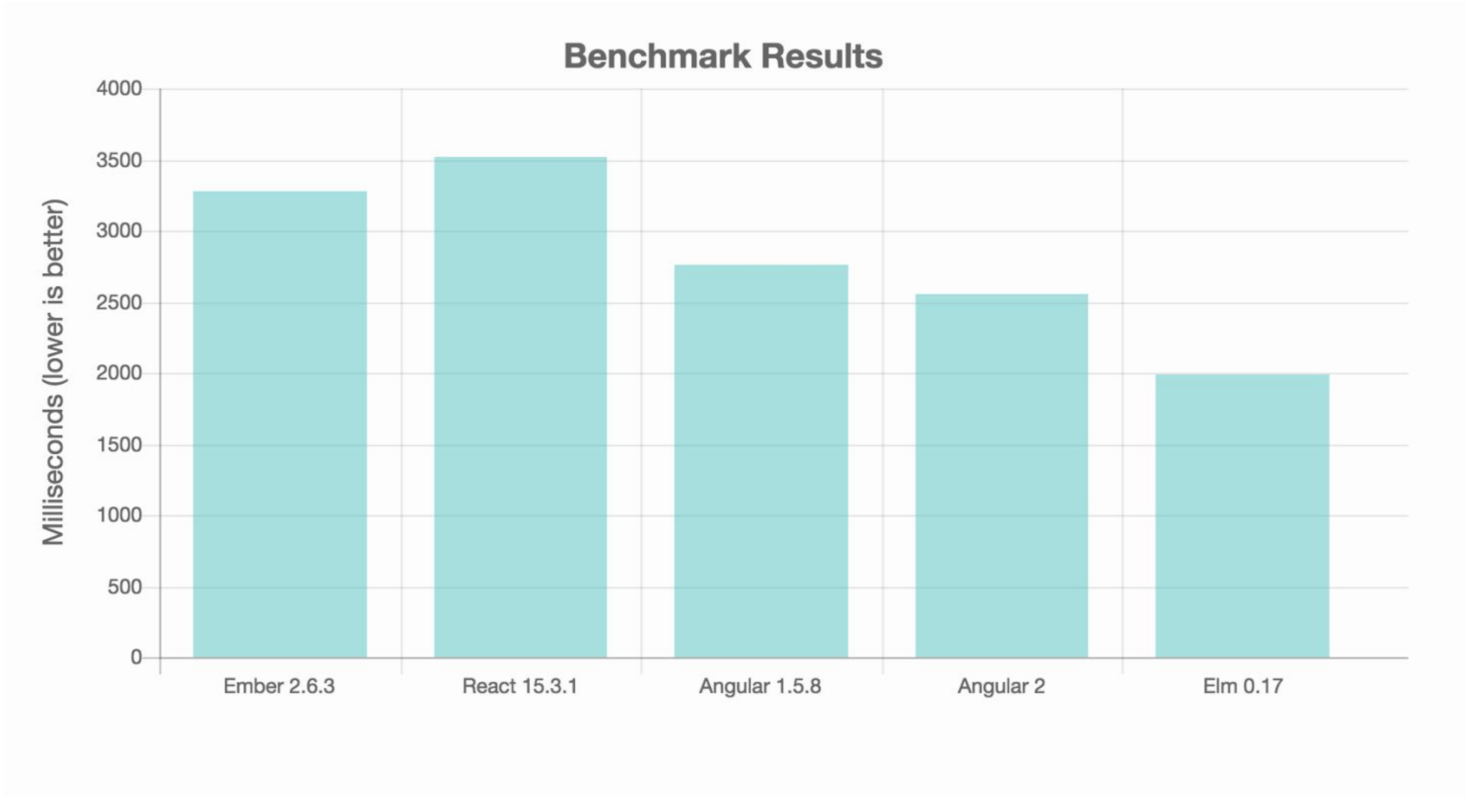
```
import Html exposing (..)
import Html.Attributes exposing (..)
import Html.Events exposing (..)
~~import Html.App as App~~

main : Program Never
main =
    ~~App.~~beginnerProgram
        { model = init
        , update = update
        , view = view
        }
```

Since 0.18

finstack

## Performance

Benchmark: Adding 100 items in the TodoMVC app



## Ecosystem

- Write HTML with elm-html
- Write CSS with elm-css
- Material Design with elm-mdl
- Make HTTP calls with elm-http
- Unit testing with elm-test
- ...

Guess the pattern? More Awesome Elm

## Learn More

- http://elm-lang.org/
- Elm Slack
- http://exercism.io/languages/elm
- Elm Weekly Newsletter
- On Twitter
  - @elmlang
  - @czaplic
  - @rtfeldman
  - @elmcastio
  - @elmstuff

finstack

# Elm vs ScalaJS

| | ScalaJS | Elm |
|---|---|---|
| **Opinionated** | No | Highly |
| **JS Interoperability** | Easy | Hard (discouraged) |
| **SPA Framework** | scalajs-react, udash.io | elm-html (TEA) |
| **Runtime Size** | 100KB | 10KB |
| **Build Tool** | SBT | elm-package |
| **Semantic Versioning** | No | Yes |
| **IDE Support** | Excellent in Intellij | Excellent in Atom |
| **Debugger** | Standard IDE tools | Time-travel debugger |
| **Ecosystem** | Huge | Small but growing fast |

## Elm in Scala Projects

SBT Elm plugin

Check out *examples* folder

Elm records generated from Scala case classes with *scala-elm-types*

# Show time!