Elm Workshop

Functional Kats February 2016

Michael Twomey

@micktwomey

What is Elm?

The best of functional programming in your browser
— elm-lang.org

- Focussed on the web front end
- Strongly Typed (in a good way)
- ML inspired
- Compiles to Javascript
- Compiler is your friend (really!)

Caveats

- Still a very young language
- Significant syntax changes still happening (though they try to minimize pain)

What I'll be Covering

1. Getting started with Elm
2. Elm basics
3. Hopefully some Elm less basics 😄

Feel free to interrupt to ask questions and correct me.

A Taste of Elm

```
module HelloWorld where

import Html exposing (h1, text, Html)
import Html.Attributes exposing (class)

main : Html
main =
  h1 [ class "welcome" ] [ text "Hello World!" ]
```

A Taste of Elm (Graphical!)

```
module HelloText where

import Graphics.Element exposing (Element, leftAligned)
import Text exposing (..)

text : Text
text =
  Text.fromString "Hello World!"
    |> bold
    |> height 24

main : Element
main = leftAligned text
```

Let's Get Started!

Some Boring Setup Bits

Install Elm

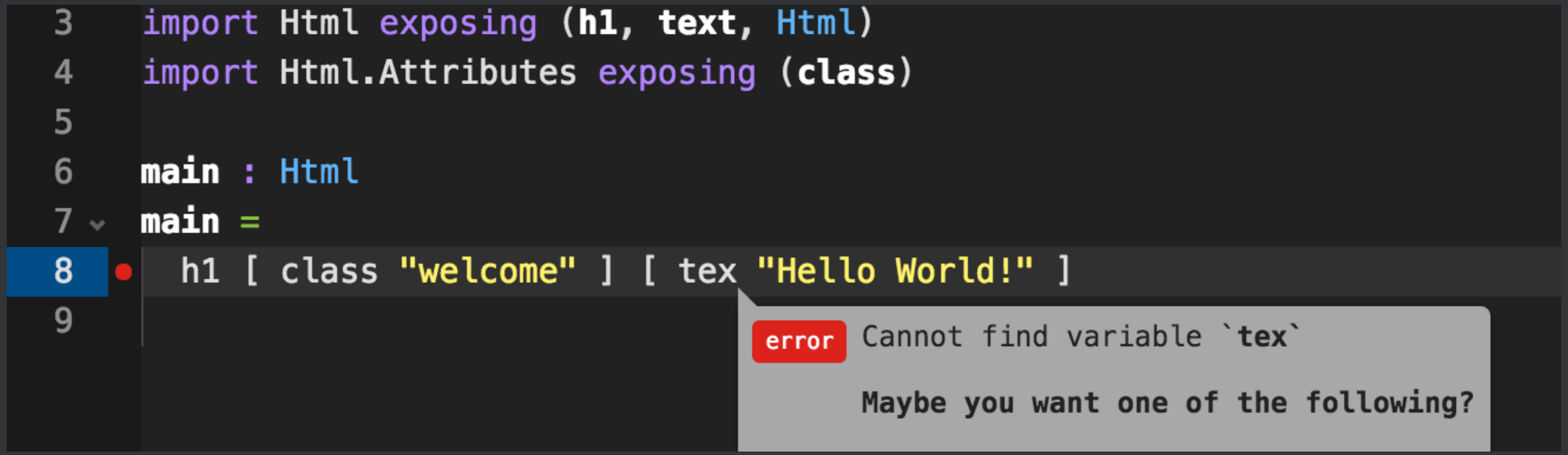
http://elm-lang.org/install

NB version 0.16

Editor Setup!

Install Atom (Controversial!)

https://atom.io



Install Plugins

language-elm

linter-elm-make

Setup Code

```
git clone https://github.com/micktwomey/elm-functionalkats-tutorial.git
cd elm-functionalkats-tutorial
elm package install
```

Digression: Package Manager (It's very polite)

```
Some new packages are needed. Here is the upgrade plan.

  Install:
    elm-lang/core 3.0.0
    evancz/elm-html 4.0.2
    evancz/virtual-dom 2.1.0

Do you approve of this plan? (y/n) y
Downloading elm-lang/core
Downloading evancz/elm-html
Downloading evancz/virtual-dom
```

Digression 2: Semantic Versioning

```
elm package diff evancz/elm-html 3.0.0 4.0.2
Comparing evancz/elm-html 3.0.0 to 4.0.2...
This is a MAJOR change.

------ Changes to module Html.Attributes - MAJOR ------

    Removed:
        boolProperty : String -> Bool -> Attribute
        stringProperty : String -> String -> Attribute


------ Changes to module Html.Events - MINOR ------

    Added:
        type alias Options =
            { stopPropagation : Bool, preventDefault : Bool }
        defaultOptions : Html.Events.Options
        onWithOptions : String -> Html.Events.Options -> Json.Decode.Decoder a -> (a -> Signal.Message) -> Html.Attribute
```

Exercise 1: Hello World Yourself!

1. Fire up elm's reactor **`elm reactor`**
2. Go to http://localhost:8000/
3. Open exercises/HelloWorldYourself.elm
4. Try running in the reactor (http://localhost:8000/exercises/HelloWorldYourself.elm) or compile with **`elm make exercises/HelloWorldYourself.elm`**
5. Let it explode and see what fun errors you get :)

HelloWorldYourself.elm

```
module HelloWorldYourself where

import Html exposing (..)
import Html.Attributes exposing (..)

main : Html
main =
  "Hello World Yourself!"
```

Solution

```
module HelloWorldYourself where

import Html exposing (..)
import Html.Attributes exposing (..)

main : Html
main =
  h1 [ class "hello" ] [ text "Hello World Yourself!" ]
```

(Yes, the class is superfluous, you can leave those brackets empty.)

Documentation

http://package.elm-lang.org

e.g. for Html.h1: http://package.elm-lang.org/packages/evancz/elm-html/4.0.2/Html#h1

Digression 3: There's a REPL

```
elm repl
```

But I never use it, the compiler is far more helpful.

Type Annotations

From Html.h1:

```
h1 : List Attribute -> List Html -> Html
```

Html.h1 takes a List of Attribute, and a List of Html and returns Html.

(Someone else can explain it better than me. Evan explained it to me and I promptly forgot.)

Exercise: HelloWorldFunction.elm

Fill in the header function in HelloWorldFunction.elm

```
module HelloWorldFunction where

import Html exposing (..)
import Html.Attributes exposing (..)

header : ?
header title =
  ?

main : Html
main =
  header "Hello World"
```

let

```
main =
  let
    pageTitle = "Hello World"
    header = h1 [] [ text pageTitle ]
  in
    header
```

Exercise: HelloLet.elm

Fill in the let expression in HelloLet.elm

Solution

```
module HelloLet where

import Html exposing (..)

header : String -> Html
header title =
  let
    fullTitle = "The title: " ++ title
    htmlTitle = text fullTitle
  in
    h1 [] [ htmlTitle ]

main : Html
main =
  header "Hello World"
```

Pipes

Nifty syntax feature every language should have

```
(height 24 (italic (fromString "Hello World!")))
```

Becomes

```
fromString "Hello World!"
|> italic
|> height 24
```

Quick Exercise: HelloPipe.elm

You know what to do, use pipes :)

```
main = leftAligned (height 24 (bold (fromString "Hello World!")))
```

Solution

```
module HelloText where

import Graphics.Element exposing (Element, leftAligned)
import Text exposing (..)

main : Element
main =
  fromString "Hello World!"
    |> bold
    |> height 24
    |> leftAligned
```

Types and Case

```
type MyType
  = Something
  | Else
  | Other String

useMyType : MyType -> String
useMyType myType =
  case myType of
    Something ->
      "Something"
    Else ->
      "Else!"
    Other string ->
      "Other: " ++ string
```

Exercise: HelloTypes.elm

Can you complete HelloTypes.elm?

Hints:

1. Start by filling out the type definition
2. The compiler error should start guiding you to the right case statements
3. You might need more helper functions
4. List.map is new, it takes a function and applies it to each item in a list.

Solution

```
type Greeting
  = Header String
  | Ul (List String)
  | Paragraph String
  | Numbers (List Int)
```

Solution Part 2

```
greeting : Greeting -> Html
greeting greet =
  case greet of
    Header header ->
      h1 [] [ text header ]
    Ul strings ->
      ul [] (List.map stringToLi strings)
    Paragraph para ->
      p [] [ text para ]
    Numbers numbers ->
      -- Ha! I cheated! I used syntax you haven't seen!
      ol [] (List.map (\x -> li [] [text (toString x) ]) numbers)
```

Signals

Go from a static app to a dynamic app

slide_examples/KeyboardSignals.elm:

```
main : Signal Element
main =
  Signal.map show arrows
```

Shows state of arrow keys.

Interlude: Signal Visualization

Excellent signal viewer: http://yang-wei.github.io/elmflux/

Exercise: ArrowPresses.elm

Render a different triangle for each arrow press.

See http://package.elm-lang.org/packages/elm-lang/core/3.0.0/Graphics-Collage

Solution

See solutions/ArrowPresses.elm

What's with those Debug.watch calls?

Bonus: Time Travel Debugging!

http://localhost:8000/solutions/ArrowPresses.elm?debug

Models and State

We've seen views and signals, the next part is state.

Records

```
type alias Model =
  { x : Int
  , y : Int
  , colour: Color
  }


model : Model
model = { x = 1, y = 2, color = Color.blue }
```

Updating Records

```
newModel = { Model
   | x = 1
   , y = 2
   , colour = Color.red
   }
```

You can't mutate a model, only return a new one.

Mind Melting Bit: foldp

```
main : Signal Element
main =
  Signal.foldp update init Keyboard.arrows
    |> Signal.map view
```

Exercise

Can you fill in the **`initPlayer`** and **`update`** functions in exercises/ArrowModel.elm?

Solution

See solutions/ArrowModel.elm

```
initPlayer : Model
initPlayer =
  { x = 0
  , y = 0
  , shape = playerShape
  }

update : { x : Int, y : Int } -> Model -> Model
update {x, y} player =
  { player
  | x = player.x + (x * 10)
  , y = player.y + (y * 10)
  }
```

Stuff I Haven't Covered

- Effects and Tasks (how you do HTTP requests)
- start-app (the canonical architecture pattern codified in a library)
- WebGL (did I mention Elm has a GL shader compiler tucked away in it?)

Further Reading

- Elm Architecture Tutorial: https://github.com/evancz/elm-architecture-tutorial/
  - My warmup tutorial: https://github.com/micktwomey/elm-tutorial
- How to Create Tetris in Elm: https://goo.gl/TgQeRy
- https://github.com/micktwomey/elm-functionalkats-tutorial