Introduction to Elm 0.16

Michael Twomey

December 8th

Goal

Get you set up and up to the start of the Elm Architecture Tutorial

https://github.com/evancz/elm-architecture-tutorial

Why Elm? Frontend

Functional Reactive Programming

Fun!

(Not Javascript! 😝)

Setting Up
(The boring bit)

Install Elm

http://elm-lang.org/install

Install Atom

https://atom.io

Install Plugins

(Remember to 🙋 for help!)

- language-elm
- linter-elm-make

Configure Plugins

- **`linter-elm-make`** might need help to find elm-make.
  - **`/usr/local/bin/elm-make`**
- Indentation: Elm style guide recommends 2 or 4, pick one and stick with it 😀
  - (I use 2 spaces)

Elm Repl

```
$ elm repl
---- elm repl 0.16.0---------------------------------------------------------
 :help for help, :exit to exit, more at <https://github.com/elm-lang/elm-repl>
-----------------------------------------------------------------------------
>
```

```
[~/s/ /e/e/counter (ma ster) $ elm repl
---- elm repl 0.16.0 ---------------------------------------------------------
 :he p for help  :exi  to exit, more at <https://github.com/elm-lang/elm-repl>
--------------------------------------------------------------------------------
>
```

NB! 0.16.0!

```
$ elm repl
---- elm repl 0.16.0 -----------------------------------------------------
 :help for help, :exit to exit, more at <https://github.com/elm-lang/elm-repl>
--------------------------------------------------------------------------
> 1 + 1
2 : number
> "Hello there"
"Hello there" : String
> [1, 2, 3]
[1,2,3] : List number
> {x = 1, y = 2}
{ x = 1, y = 2 } : { x : number, y : number' }
```

Hello World

```
mkdir hello
cd hello
atom .
elm package install evancz/elm-html
touch Hello.elm
elm reactor
```

If you go to http://localhost:8000 and click on Hello.elm you'll get a compiler error. This is good.

Hello.elm

```elm
import Html


-- View
main : Html.Html
main =
    -- Deliberate mistake here, to show off errors
    Html.h1 [] [ "Hello World" ]
    -- Html.h1 [] [ Html.text "Hello World" ]
```

Hello.elm

```
1  import Html
2
3  -- View
4  main : Html.Html
5  main =
6    -- Deliberate mistake here, to show off errors
7    Html.h1 [] [ "Hello World" ]
8    -- Html.h1 [] [ Html.text "H
9
```

error The 2nd argument to function `h1` is causing a mismatch.

Function `h1` is expecting the 2nd argument to be:

    List VirtualDom.Node

But it is:

    List String

Hint: I always figure out the type of arguments from left to right. If an argument is acceptable when I check it, I assume it is "correct" in subsequent checks. So the problem may actually be in how previous arguments interact with the 2nd.

```elm
import Html


-- View
main : Html.Html
main =
    Html.h1 [] [ Html.text "Hello World" ]
```

http://localhost:8000/Hello.elm

Slightly less typing

```
import Html exposing (..)


-- View
main : Html
main =
    h1 [] [ text "Hello World" ]
```

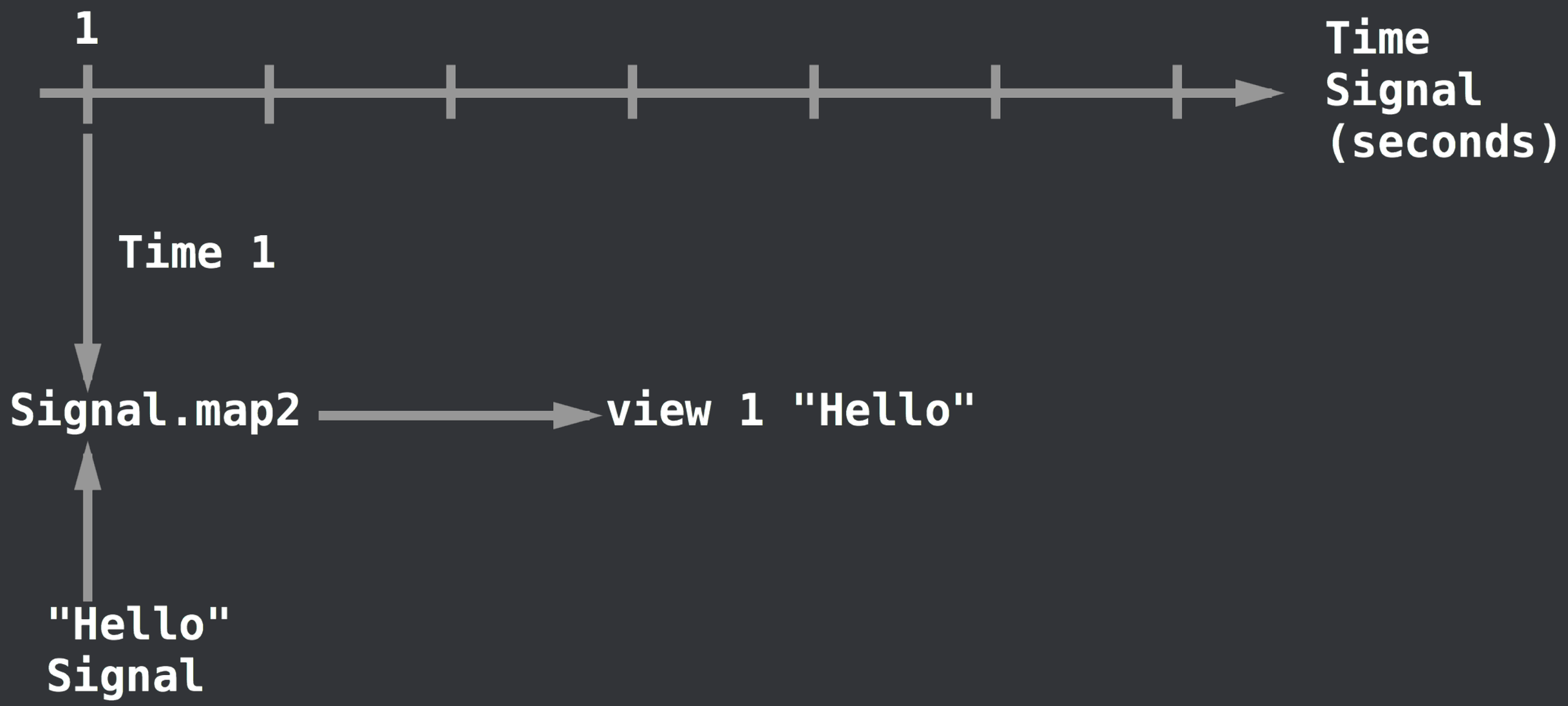
Functions

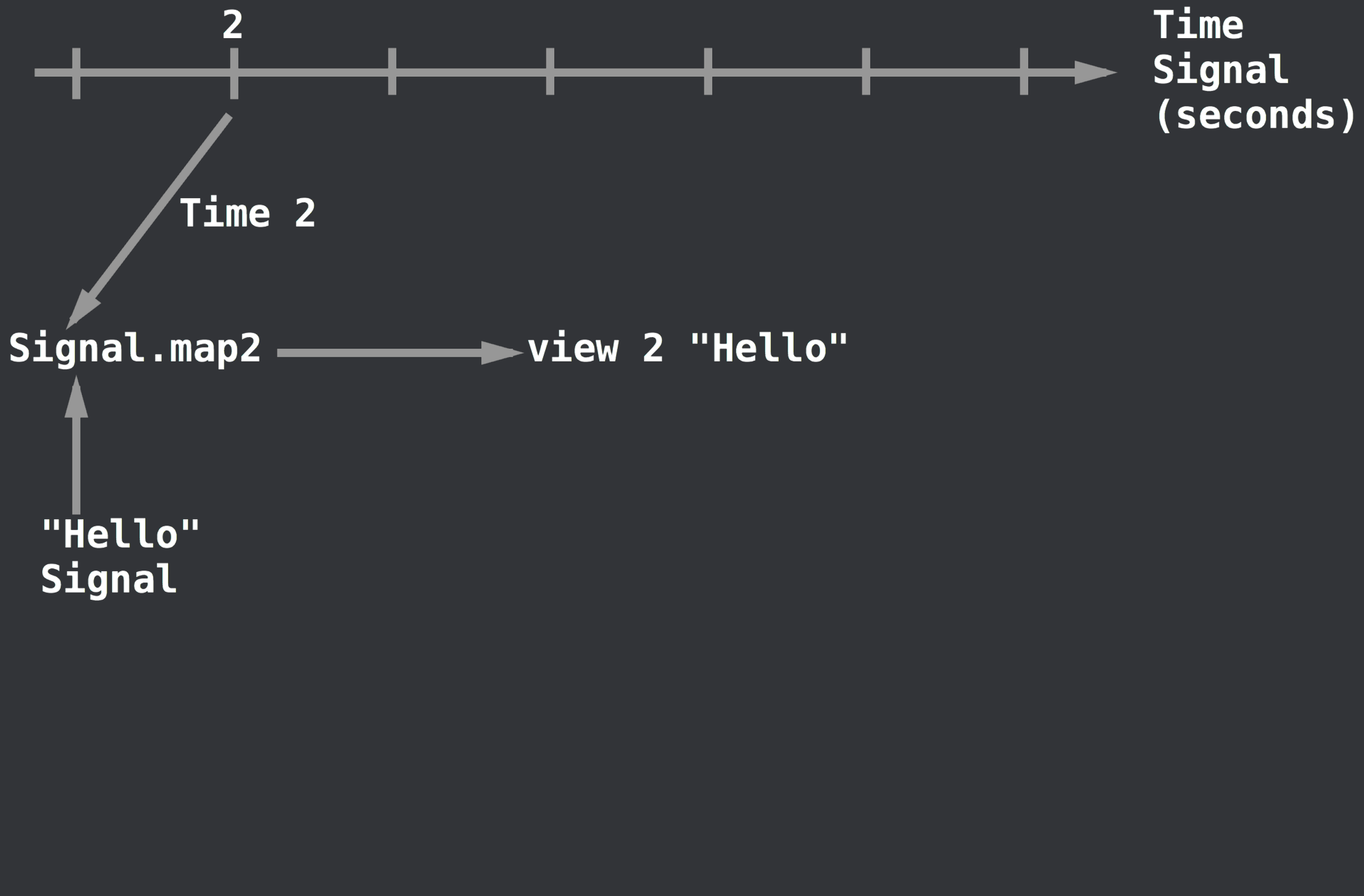
```
import Html exposing (..)


-- View
view : String -> Html
view message =
  h1 [] [ text message ]


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

```elm
import Html exposing (..)
import Time

-- View

view : Float -> String -> Html
view time message =
  h1 [] [ text (message ++ toString time) ]

main : Signal Html
main =
  Signal.map2 view (Time.every Time.second) (Signal.constant "Hello: ")
```


```
Signal.map2                    view
```

```
"Hello"
Signal
```


1
Time
Signal
(seconds)
Time 1
Signal.map2
view 1 "Hello"
"Hello"
Signal

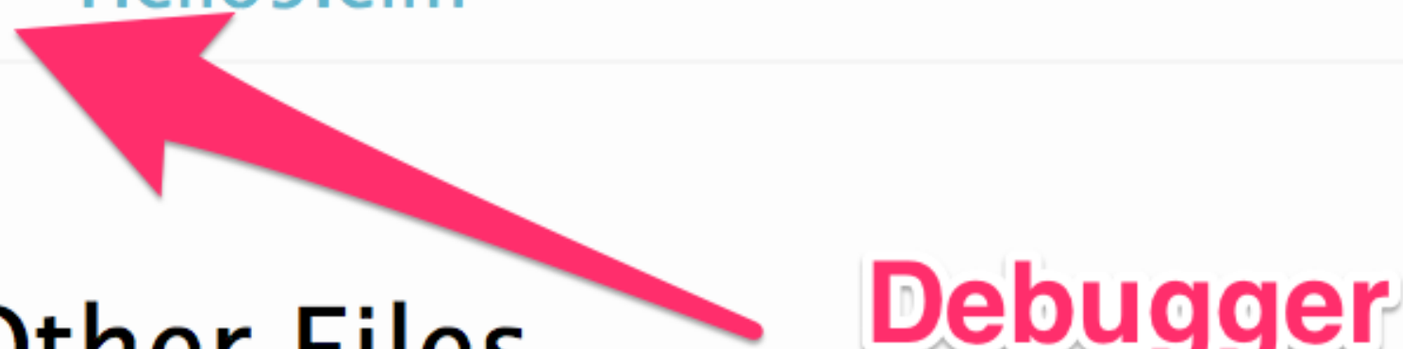

2
Time
Signal
(seconds)
Time 2
Signal.map2
view 2 "Hello"
"Hello"
Signal

```
import Html exposing (..)
import Time

-- View

view : Float -> String -> Html
view time message =
  let
    full_message = message ++ toString time
  in
    h1 [] [ text full_message ]

main : Signal Html
main =
  Signal.map2 view (Time.every Time.second) (Signal.constant "Hello: ")
```

```
import Debug
import Html exposing (..)
import Time

-- View

view : Float -> String -> Html
view time message =
  let
    full_message = message ++ toString time
    _ = Debug.watch "full_message is" full_message
  in
    h1 [] [ text full_message ]

main : Signal Html
main =
  Signal.map2 view (Time.every Time.second) (Signal.constant "Hello: ")
```

Elm Files

- Hello.elm
- Hello2.elm
- Hello3.elm
- Hello4.elm
- Hello5.elm



Other Files

Hello: 14493131143378

swap

10

0 10

full_message is

"Hello: 14493131143378"

Covered so far

- Elm Reactor
- Importing
- Basic functions
- Function Signatures
- Signals
- Let Expressions
- Debugger
- "Views"

Bonus: Prettier dates

```
elm package install mgold/elm-date-format
```

```elm
import Date
import Date.Format exposing (format)
import Debug
import Html exposing (..)
import Time

-- View

view : Float -> String -> Html
view time message =
  let
    _ = Debug.watch "raw time is" time
    full_message = message ++ format "%A %H:%m:%S" (Date.fromTime time)
    _ = Debug.watch "full_message is" full_message
  in
    h1 [] [ text full_message ]

main : Signal Html
main =
  Signal.map2 view (Time.every Time.second) (Signal.constant "Hello: ")
```

Counter

First part of the Elm Architecture Tutorial covers counters, so we're gonna create one. Getting there...

```
cd ..
mkdir counter
cd counter
elm package install evancz/elm-html
elm package install evancz/start-app
atom .
touch Counter.elm Main.elm
elm reactor
```

Counter.elm

```
module Counter where


-- Model


type alias Model = Int
```

```elm
module Counter where
import Html exposing (..)
import Html.Attributes exposing (style)

-- Model
type alias Model = Int

-- Views
view : Model -> Html
view model =
  div []
    [ button [] [ text "-" ]
    , div [ style [("font-size", "20px")] ] [ text (toString model) ]
    , button [] [ text "+" ]
    ]

main : Html
main =
  view 0
```

```
-- View
countStyle : Attribute
countStyle =
  style
    [ ("font-size", "20px")
    , ("font-family", "monospace")
    , ("display", "inline-block")
    , ("width", "50px")
    , ("text-align", "center")
    ]

view : Model -> Html
view model =
  div []
    [ button [] [ text "-" ]
    , div [ countStyle ] [ text (toString model) ]
    , button [] [ text "+" ]
    ]
```

Main.elm

```
import Counter exposing (update, view)

import Html exposing (Html)
import StartApp.Simple exposing (start)

main : Signal Html
main =
  start
    { model = 0
    , update = update
    , view = view
    }
```

```
...
-- Model
type alias Model = Int

-- Update
type Action =
  Increment
  | Decrement

update : Action -> Model -> Model
update action model =
  0

-- View
...
-- Delete main : Html at end of file
```

```
...
import Html.Attributes exposing (style)
import Html.Events exposing (onClick)
...
update : Action -> Model -> Model
update action model =
  case action of
    Increment ->
      model + 1
    Decrement ->
      model - 1
...
view : Signal.Address Action -> Model -> Html
view address model =
  div []
    [ button [ onClick address Decrement ] [ text "-" ]
    , div [ countStyle ] [ text (toString model) ]
    , button [ onClick address Increment ] [ text "+" ]
    ]
```

Woo, a Counter!

Elm Learning

- http://elm-lang.org/docs
- https://github.com/evancz/elm-architecture-tutorial/
- http://package.elm-lang.org
- https://pragmaticstudio.com/elm
- https://pragmaticstudio.com/elm-signals