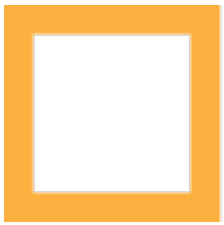
Writing applications in Elm
Functional Programming

Jens Egholm Pedersen and Anders Kalhauge



cphbusiness

Spring 2017

Outline

Railroad oriented programming

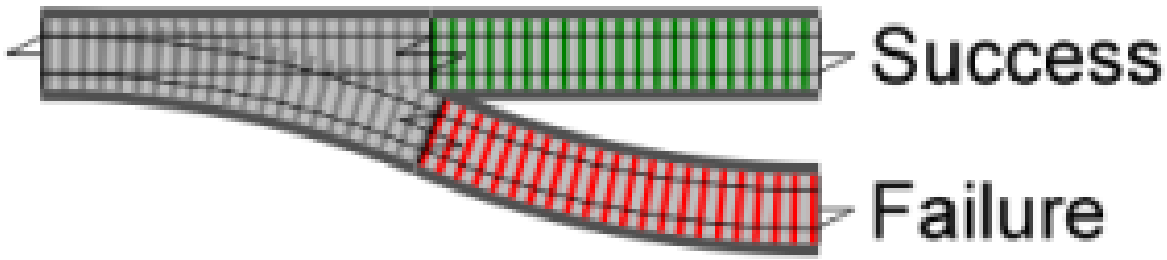
Parsing JSON in Elm
- Decoder pipeline
- Exercise 1

Subscriptions
- Exercise 2

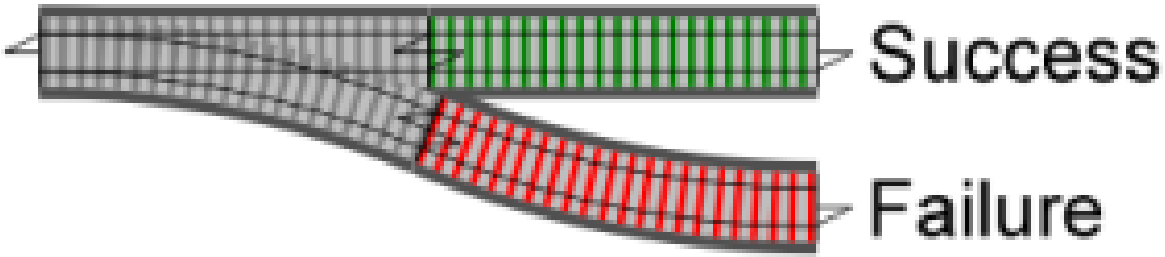
Websockets
- Exercise 3

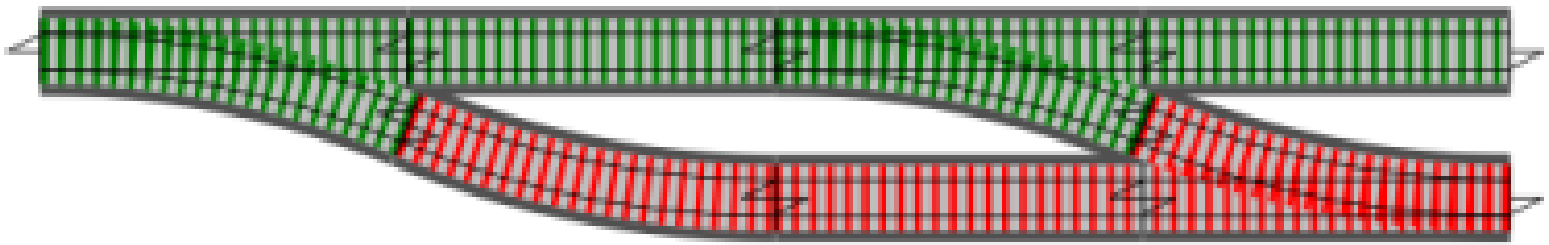
Websocket assignment

Railway



Railway





Pipeline: A sequence of functions chained together.

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Pipeline: A sequence of functions chained together. **Pipe**: Uses the output of one function as the input to another.

```
(|>) : a -> (a -> b) -> b
```

S

Pipeline: A sequence of functions chained together. **Pipe**: Uses the output of one function as the input to another.

```
(|>) : a -> (a -> b) -> b
```

S

```
function 1 |> function 2
```

Pipeline: A sequence of functions chained together. **Pipe**: Uses the output of one function as the input to another.

```
(|>) : a -> (a -> b) -> b
```

s

```
function 1 |> function 2
```

```
function 1 |> function 2 |> ... |> function n
```

Pipes example

```
["Daniel", "Dennett"]
  |> String.join ""
  |> String.length -- 13
```

JSON parsing in Elm

JSON parsing

- What is the input?

JSON parsing

- What is the input?

  `String`

JSON parsing

cphbusiness
COPENHAGEN BUSINESS ACADEMY

- What is the input?

  `String`
- What is the output?

JSON parsing

cphbusiness
COPENHAGEN BUSINESS ACADEMY

- What is the input?

  ```
  String
  ```

- What is the output?

  ```
  Result Http.Error String
  ```

JSON decoders

cphbusiness
COPENHAGEN BUSINESS ACADEMY

From the package `Decode`

JSON decoders

cphbusiness
COPENHAGEN BUSINESS ACADEMY

From the package `Decode`

A decoder decodes to a type a

JSON decoders

cphbusiness
COPENHAGEN BUSINESS ACADEMY

From the package `Decode`

A decoder decodes to a type a

```
Decoder a
```

JSON decoders

From the package Decode

A decoder decodes to a type a

```
Decoder a
```

```
Decode.int -- simply decodes JSON int to Elm Int
```

Running Elm decoder

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
decodeString : Decoder a -> String -> Result String a
```

Running Elm decoder

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
decodeString : Decoder a -> String -> Result String a
```

```
decodeString int "4"       == Ok 4
decodeString int "1 + 2"   == Err ...
```

Combining decoders


```
list : Decoder a -> Decoder (List a)
```

Combining decoders


```
list : Decoder a -> Decoder (List a)
```

```
list int -- Decoder (List Int)
```

Decoding object fields

```
jsonString : String
jsonString = """{ "name": "David Chalmers"}"""
```

Decoding object fields

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString = """{ "name": "David Chalmers"}"""
```

```
field : String -> Decoder a -> Decoder a
```

Decoding object fields

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString = """{ "name": "David Chalmers"}"""
```

```
field : String -> Decoder a -> Decoder a
```

```
at "name" string
```

Decoding nested objects

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString =
  """{ "result": { "name": "David Chalmers"}}"""
```

Decoding nested objects

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString =
  """{ "result": { "name": "David Chalmers"}}"""
```

```
at : List String -> Decoder a -> Decoder a
```

Decoding nested objects

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString =
  """{ "result": { "name": "David Chalmers"}}"""
```

```
at : List String -> Decoder a -> Decoder a
```

```
at ["result", "name"] string
```

Decoding nested objects

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString =
  """{ "result": { "name": "David Chalmers"}}"""
```

Decoding nested objects

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString =
  """{ "result": { "name": "David Chalmers"}}"""
```

```
at : List String -> Decoder a -> Decoder a
```

Decoding nested objects

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString =
  """{ "result": { "name": "David Chalmers"}}"""
```

```
at : List String -> Decoder a -> Decoder a
```

```
at ["result", "name"] string
```

Decoding arrays

```
jsonString : List String
jsonString = """[␣"Donald",␣"Davidson"␣]"""
```

Decoding arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : List String
jsonString = """[␣"Donald",␣"Davidson"␣]"""
```

```
index : Int -> Decoder a -> Decoder a
```

Decoding arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : List String
jsonString = """[ "Donald", "Davidson" ]"""
```

```
index : Int -> Decoder a -> Decoder a
```

```
index 0 string
```

JSON parsing example

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString = """{ "name": "Richard Dawkins" }"""
```

JSON parsing example

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString = """{ "name": "Richard Dawkins" }"""
```

```
type alias Person
 = { name: String }
```

JSON parsing example

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString = """{ "name": "Richard Dawkins" }"""
```

```
type alias Person
 = { name: String }
```

```
personParser : Decoder Person
personParser =
  field "name" string
```

JSON parsing example

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
jsonString : String
jsonString = """{ "name": "Richard Dawkins" }"""
```

```
type alias Person
 = { name: String }
```

```
personParser : Decoder Person
personParser =
  field "name" string
```

```
parsePerson : String -> Result String Person
parsePerson =
  decodeString personParser
```

NoRedInk/elm-decode-pipeline

> *A library for building decoders using the pipeline (|>) operator and plain function calls.*

Decoder pipeline

NoRedInk/elm-decode-pipeline

> *A library for building decoders using the pipeline (|>) operator and plain function calls.*

Install with `elm-package install NoRedInk/elm-decode-pipeline`

Building a pipeline

Building a pipeline:

Building a pipeline

Building a pipeline:

```
decode : a -> Decoder a
```

Building a pipeline

Building a pipeline:

```
decode : a -> Decoder a
```

```
(|>) : a -> (a -> b) -> b
```

Building a pipeline

Building a pipeline:

```
decode : a -> Decoder a
```

```
(|>) : a -> (a -> b) -> b
```

Required fields:

Building a pipeline

Building a pipeline:

```
decode : a -> Decoder a
```

```
(|>) : a -> (a -> b) -> b
```

Required fields:

```
required : String -> Decoder a
        -> Decoder (a -> b) -> Decoder b
```

Building a pipeline

Building a pipeline:

```
decode : a -> Decoder a
```

```
(|>) : a -> (a -> b) -> b
```

Required fields:

```
required : String -> Decoder a
        -> Decoder (a -> b) -> Decoder b
```

Optional fields:

Building a pipeline

Building a pipeline:

```
decode : a -> Decoder a
```

```
(|>) : a -> (a -> b) -> b
```

Required fields:

```
required : String -> Decoder a
        -> Decoder (a -> b) -> Decoder b
```

Optional fields:

```
optional : String -> Decoder a
        -> a -> Decoder (a -> b) -> Decoder b
```

Decoder pipeline example 1/2

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
import Json.Decode.Pipeline exposing (..)
```

Decoder pipeline example 1/2

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
import Json.Decode.Pipeline exposing (..)
```

```
type alias User
  = { id : Int }
```

Decoder pipeline example 1/2

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
import Json.Decode.Pipeline exposing (..)
```

```
type alias User
  = { id : Int }
```

```
userDecoder : Decoder User
userDecoder =
  decode User
    |> required "id" int
```

Decoder pipeline example 2/2

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
import Json.Decode.Pipeline exposing (..)
```

```
type alias User
  = { id : Int
    , name : String
    , email : String
    }
```

Decoder pipeline example 2/2

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
import Json.Decode.Pipeline exposing (..)
```

```
type alias User
  = { id : Int
    , name : String
    , email : String
    }
```

```
userDecoder : Decoder User
userDecoder =
  decode User
    |> required "id" int
    |> required "name" string
    |> optional "email" string "no email"
```

Exercise 1: Parse JSON

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Clone the `elm-exercises` from `cphbus-functional-programming`

`https://github.com/cphbus-functional-programming/elm-exercises`

Work on the `json.elm` file in the `basicelm` folder

Goal 1: Parse the incoming JSON to a `Person`

Goal 2: Display all the fields in the `Person` type in the HTML

Handled by the `Time` package

Time in Elm

Handled by the `Time` package

```
type alias Time = Float
```

Converting time in Elm

`Time` can be converted into

- Hours: `inHours : Time -> Float`

Converting time in Elm

cphbusiness
COPENHAGEN BUSINESS ACADEMY

`Time` can be converted into

- Hours: `inHours : Time -> Float`
- Minutes: `inMinutes : Time -> Float`

Converting time in Elm

`Time` can be converted into

- Hours: `inHours : Time -> Float`
- Minutes: `inMinutes : Time -> Float`
- ... and seconds and milliseconds

Periodic updates

Periodic updates is something in a fixed interval

Periodic updates

Periodic updates is something in a fixed interval

Like `setInterval` in JavaScript

Periodic updates

Periodic updates is something in a fixed interval

Like `setInterval` in JavaScript

```
every : Time -> (Time -> msg) -> Sub msg
```

Periodic updates

Periodic updates is something in a fixed interval

Like `setInterval` in JavaScript

```
every : Time -> (Time -> msg) -> Sub msg
```

`msg` is used in `update`

Periodic updates

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Periodic updates is something in a fixed interval

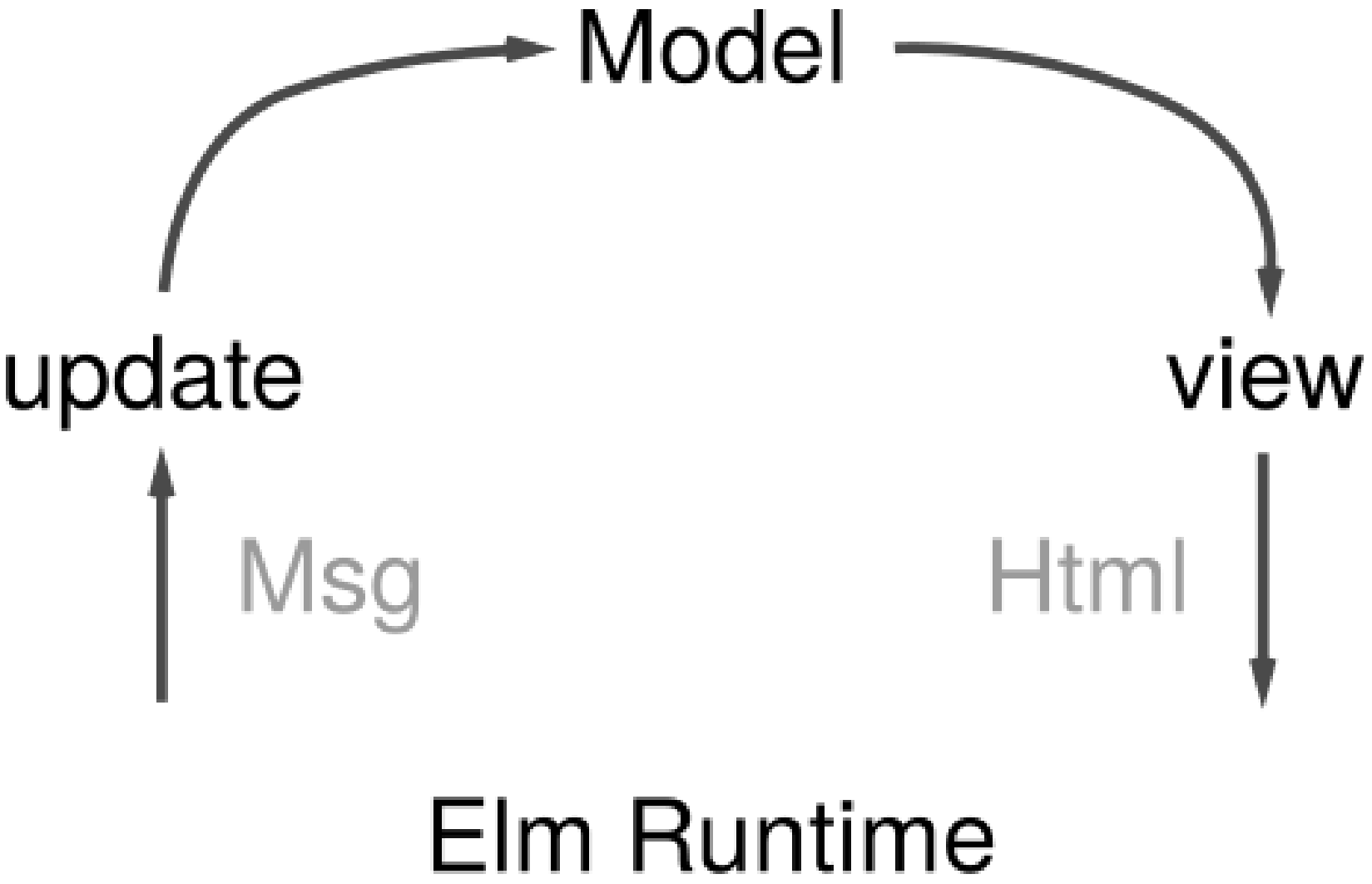
Like `setInterval` in JavaScript

```
every : Time -> (Time -> msg) -> Sub msg
```

`msg` is used in `update`

```
update : msg -> model -> model
```

HTML updates



Subscriptions in Elm

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
type Sub msg
```

Subscriptions in Elm

```
type Sub msg
```

A subscription is a way of telling Elm, "Hey, let me know if anything interesting happens over there!"

Subscriptions in Elm

```
type Sub msg
```

> *A subscription is a way of telling Elm, "Hey, let me know if anything interesting happens over there!"*

What's the Input?

Subscriptions in Elm

```
type Sub msg
```

> *A subscription is a way of telling Elm, "Hey, let me know if anything interesting happens over there!"*

What's the Input? And output?

Subscriptions in Elm

```
type Sub msg
```

> *A subscription is a way of telling Elm, "Hey, let me know if anything interesting happens over there!"*

What's the Input? And output?

```
type Msg = Tick Time
```

Subscriptions in Elm

```
type Sub msg
```

> *A subscription is a way of telling Elm, "Hey, let me know if anything interesting happens over there!"*

What's the Input? And output?

```
type Msg = Tick Time
```

```
every : Time -> (Time -> msg) -> Sub msg
```

Subscriptions in Elm

```
type Sub msg
```

> *A subscription is a way of telling Elm, "Hey, let me know if anything interesting happens over there!"*

What's the Input? And output?

```
type Msg = Tick Time
```

```
every : Time -> (Time -> msg) -> Sub msg
```

```
subscriptions : Model -> Sub Msg
subscriptions model =
  Time.every millisecond Tick
```

Exercise 2: Timed progress bar

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Clone the `elm-exercises` from `cphbus-functional-programming`

`https://github.com/cphbus-functional-programming/elm-exercises`

Work on the `subscriptions.elm` file in the `basicelm` folder

Goal 1: Start a subscription every millisecond

Goal 2: Update the model when the subscription arrive in the update functions

Goal 3: Set the width of the second progress-bar in the view (by correctly updating the 'progress' variable in line 51) to go from 0 to 100 once every 5 seconds.

Websockets

Websockets

Defined with the URI prefix: `ws` and `wss` (secure).

Websockets

Defined with the URI prefix: `ws` and `wss` (secure).

Full duplex between over a single `TCP` channel.

Websockets

Defined with the URI prefix: `ws` and `wss` (secure).

Full duplex between over a single `TCP` channel.

What are the pros and cons of using websockets?

Websockets in Elm


```
listen : String -> (String -> msg) -> Sub msg
```

Websockets in Elm


```
listen : String -> (String -> msg) -> Sub msg
```

```
type Msg = Echo String | ...

subscriptions model =
  listen "ws://echo.websocket.org" Echo
```

Exercise 3: Websocket

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Clone the `elm-exercises` from `cphbus-functional-programming`

`https://github.com/cphbus-functional-programming/elm-exercises`

Work on the `chat.elm` file in the `chat` folder

Goal 1: Start the node websocket server available in `server.js`

Goal 2: Connect to the websocket server on port 3000 from Elm

Goal 3: Write the response from the server to your HTML

Websocket chat server assignment

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Use the following JSON protocol
{ command: "login|send", content: "userName|message" }
and extend the `chat.elm` page to:

1. Send and receive messages in `JSON`
2. Display a full list of the last chat messages
   This includes changing the model to contain a list of chat messages
3. Login with a user name
   Implement this either by prefixing the user name in the button with "login" (so to log in with "Anders" you write "loginAnders"), or by creating a new input field

Websocket chat server assignment

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Use the following JSON protocol
{ command: "login|send", content: "userName|message" }
and extend the `chat.elm` page to:

1. Send and receive messages in `JSON`
2. Display a full list of the last chat messages
   This includes changing the model to contain a list of chat messages
3. Login with a user name
   Implement this either by prefixing the user name in the button with "login" (so to log in with "Anders" you write "loginAnders"), or by creating a new input field

Hand in before **10th April 23:59**