Lab 07 - The Full Elm Architecture

CS 1XA3

Feb 26, 2018

Review: Basic Elm Architecture

```
type alias Model = ..
type Msg = ...

update : Msg -> Model -> Model
update msg model = ..

view : Model -> Html.Html Msg
view model = ...

main : Program Never Model Msg
main = Html.beginnerProgram
         { model = init,
           view = view,
           update = update }
```

Note: the use of Html.beginnerProgram

The Full Elm Architecture

```
main : Program Never Model Msg
main = program
        { init = init
        , view = view
        , update = update
        , subscriptions = subscriptions
        }
```

Two new concepts

- Commands
- Subscriptions

Changes to Update Function

- Example: BasicProgram Update

```
update : Msg -> Model -> Model
update msg model = let
       newmodel = ...
   in newmodel
```

- Example: Program Update

```
update : Msg -> Model -> ( Model, Cmd Msg )
update msg model = let
      newmodel = ...
  in (newmodel,Cmd.none)
```

Note: use Cmd.none in absence of an actual Command

Subscriptions

Subscriptions allow you to listen for external output such as general Keyboard and Mouse events (and other more complicated things we won't cover)

Example

```
type Msg = MouseMsg Mouse.Position
         | KeyMsg Keyboard.KeyCode

subscriptions : Model -> Sub Msg
subscriptions model =
    Sub.batch
        [ Mouse.clicks MouseMsg
        , Keyboard.downs KeyMsg
        ]
```

Keyboard Events

See http://package.elm-lang.org/packages/elm-lang/keyboard/1.0.1/Keyboard#

- Keys are identified by an Integer code

```
type alias Keycode = Int
```

- Convert to and from KeyCodes with

```
Char.toCode   : Char -> KeyCode
Char.fromCode : KeyCode -> Char
```

- Keyboard events are listened for by the following subscriptions

```
presses : (KeyCode -> msg) -> Sub msg
downs   : (KeyCode -> msg) -> Sub msg
ups     : (KeyCode -> msg) -> Sub msg
```

Case Study: Keyboard Events

Implement a counter that increments on the 'i' key and decrements on the 'd' key with the following code by writing an update function that case matches on KeyMsg

```
import Html as Html
import Keyboard as Key

type alias Model = { counter : Int }
type Msg = KeyMsg Key.KeyCode

view : Model -> Html.Html Msg
view model =  Html.text (toString model.counter)

subscriptions : Model -> Sub Msg
subscriptions model = Key.downs KeyMsg
```

Mouse Events

See http://package.elm-lang.org/packages/elm-lang/mouse/1.0.1/Mouse#Position

- Mouse positions are represented with a record type

```
type alias Position =  { x : Int
                       , y : Int}
```

- Mouse events take a function for transforming a position to a Msg

```
clicks : (Position -> msg) -> Sub msg
moves : (Position -> msg) -> Sub msg
downs : (Position -> msg) -> Sub msg
ups : (Position -> msg) -> Sub msg
```

Challenge: Mouse Events

Create a mouse tracker (i.e display the current position of the mouse at any time) using the following model

```
type alias Model = { position : (Int,Int) }

init : (Model,Cmd.Cmd Msg)
init = ({ position = (0,0) }, Cmd.none)

type Msg = MouseMsg Mouse.Position
```

Batching Subscriptions

- You can combine multiple Subscriptions using Platform.Sub.batch (you can also map)

```
map : (a -> msg) -> Sub a -> Sub msg
batch : List (Sub msg) -> Sub msg
```

- Example

```
type Msg = MouseMsg Mouse.Position
         | KeyMsg Keyboard.KeyCode

subscriptions : Model -> Sub Msg
subscriptions model =  Sub.batch
        [ Mouse.clicks MouseMsg
        , Keyboard.downs KeyMsg]
```

Commands

Commands can be used to execute things that involve side effects, for example

- Random Number Generation
- Http requests
- Saving to local storage

Http requests (which take advantage of server functionality) are their most common use, but since we're sticking to front-end developement we'll go over random number generation

Random

See http://package.elm-lang.org/packages/elm-lang/core/5.1.1/Random

- Generate a random value with Random.generate

```
generate : (a -> msg) -> Generator a -> Cmd msg
```

- You can provide the following generators

```
bool  : Generator Bool
int   : Int -> Int -> Generator Int
float : Float -> Float -> Generator Float
```

- Example: randomly changes the model to True or False

```
type alias Model = Bool
type Msg = Flip | Result Bool
update msg model = case msg of
    Flip -> (model,generate Result bool)
    Result b -> (b,Cmd.none)
```