Lab 05 - The Elm Architecture

CS 1XA3

Feb 6, 2018

Overview: Elm Architecture

Basic Idea

- Model: define a data type to represent the state of your program
- View: define a function that takes your Model and generates HTML
- Update: define a function that takes your current Model and generates a new one

Elm Main Program

An Elm main function generates a Program, for example

```
init   = -- instance of model type
view   = -- function generating html
update = -- function updating model

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

Types of Elm Programs

```
-- for simple applications
beginnerProgram
    :  { model : model,
         view : model -> Html msg,
         update : msg -> model -> model }
    -> Program Never model msg

-- add's some more advance functionality
-- (i.e Commands, Subscriptions)
program
    :  { init : (model, Cmd msg),
         update : msg -> model -> (model, Cmd msg),
         subscriptions : model -> Sub msg,
         view : model -> Html msg }
    -> Program Never model msg
```

Elm Record Types

- There's an additional way to define types in Elm known as Records
- Example

```
type alias Model = { someInt : Int
                   , someStr : String }

example : Model
example = { someInt = 0, someStr = "" }

increment : Model -> Model
increment model =
    { model | someInt = model.someInt + 1 }
```

Defining a Model

- Your Model can be any type or type alias you choose to define
- Usually, it's best to make a model with Record Syntax
- Example

```
type alias Model = { counter : Int }

model : Model
model = { counter = 0 }
```

Defining an Update Function

- Start by defining a Message type your update function will recieve

```
type Msg = Increment | Nothing
```

- The update function will be called with a value of Msg and the current Model

```
update :: Msg -> Model -> Model
update msg model1 = case msg of
      Increment -> { model1 |
                     counter = model1.counter + 1 }
      Nothing   -> model1
```

- The update function is triggered by certain other functions like ones in Html.Event

Defining a View Function

- You can consider the view function to work similarly to how games render graphics by frame
- The function takes the current model and returns Html, that might make use of event functions that pass a Msg to update
- Example

```
view : Model -> Html Msg
view model =
    div []
        [ div [] [ text (toString model.counter) ]
        , button [ onClick Increment ] [ text "+" ]
        ]
```

Putting it all Together

- Add the proper imports, something like

```
import Html exposing (..)
import Html.Events exposing (onClick)
```

- Then try running the code with elm-reactor