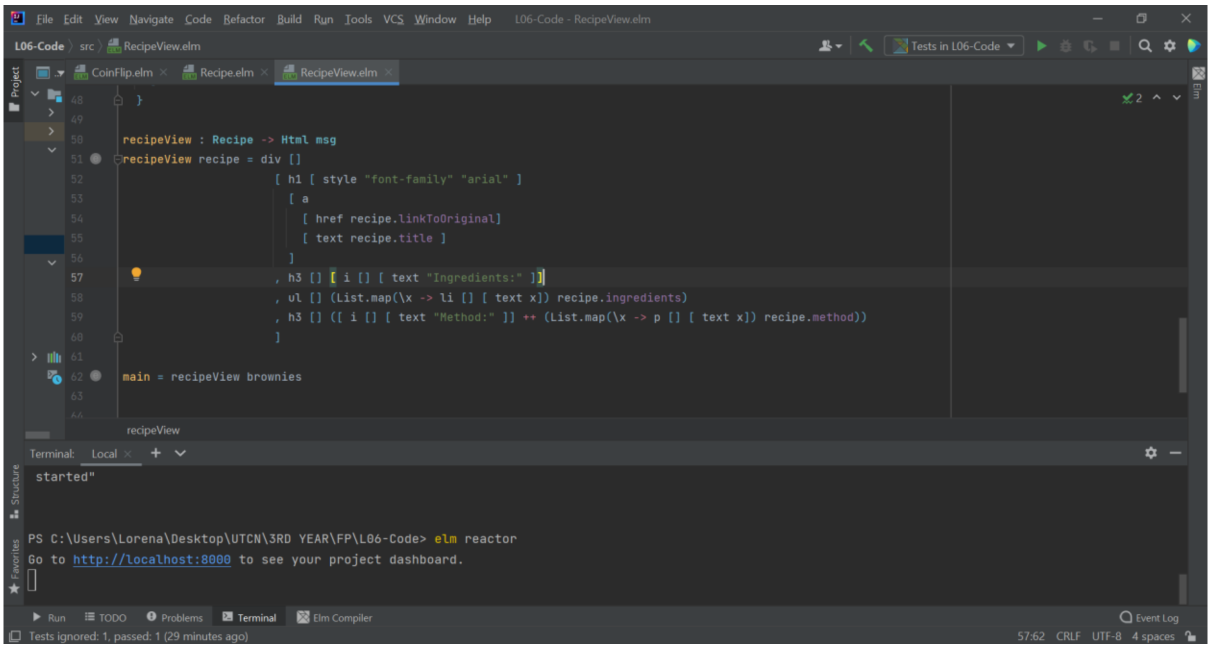
Laboratory 6

Exercise 6.2.1: Starting from the code above and the type definition for *Recipe*, write a function *recipeView : Recipe -> Html msg* that can render any recipe (i.e. avoid hardcoding the recipe data into the view).



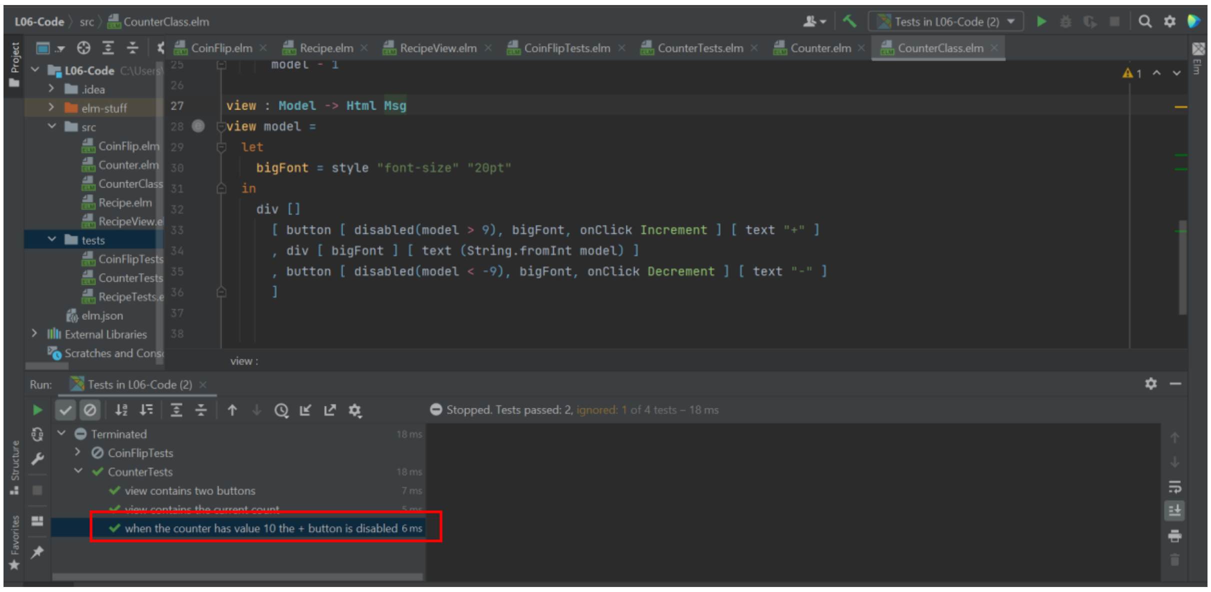
```
    }

recipeView : Recipe -> Html msg
recipeView recipe = div []
                    [ h1 [ style "font-family" "arial" ]
                      [ a
                        [ href recipe.linkToOriginal]
                        [ text recipe.title ]
                      ]
                    , h3 [] [ i [] [ text "Ingredients:" ]]
                    , ul [] (List.map(\x -> li [] [ text x]) recipe.ingredients)
                    , h3 [] ([ i [] [ text "Method:" ]] ++ (List.map(\x -> p [] [ text x]) recipe.method))
                    ]

main = recipeView brownies
```

```
started"

PS C:\Users\Lorena\Desktop\UTCN\3RD YEAR\FP\L06-Code> elm reactor
Go to http://localhost:8000 to see your project dashboard.
```

Tests ignored: 1, passed: 1 (29 minutes ago)

Exercise 6.2.2: Modify the Counter app to prevent the counter from going over 10 or under -10 by disabling the + or - buttons when the value is reached. Remove the call to skip in the CounterTests.elm file to test your implementation.

I used the disabled attribute with the 2 specific conditions (model > 9 or model < -9). Initially I tried with model > 10 and model < -10 but I observed that the test failed and the buttons were disabled only when they passed the limit of 11 and -11.

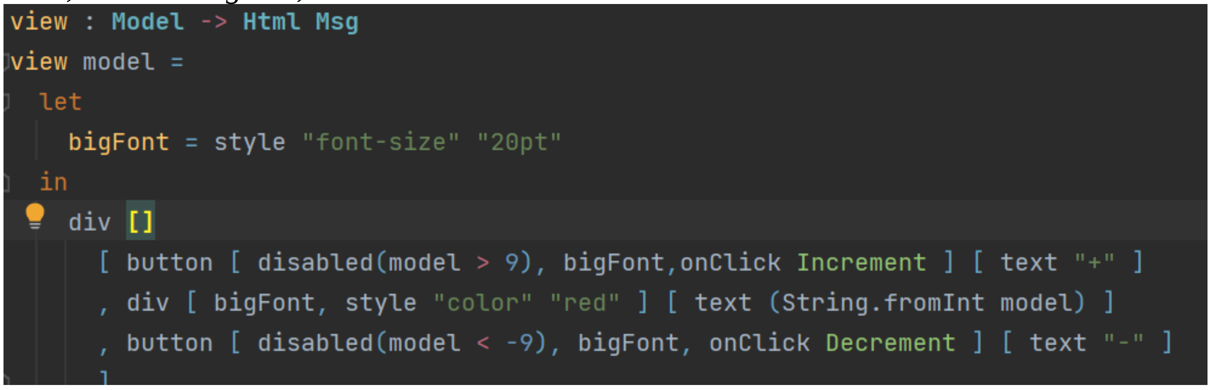


```
      model - 1

view : Model -> Html Msg
view model =
  let
    bigFont = style "font-size" "20pt"
  in
    div []
      [ button [ disabled(model > 9), bigFont, onClick Increment ] [ text "+" ]
      , div [ bigFont ] [ text (String.fromInt model) ]
      , button [ disabled(model < -9), bigFont, onClick Decrement ] [ text "-" ]
      ]
```

```
Stopped. Tests passed: 2, ignored: 1 of 4 tests – 18 ms
Terminated                                              18 ms
  CoinFlipTests
  CounterTests                                          18 ms
    view contains two buttons                            7 ms
    view contains the current count                      5 ms
    when the counter has value 10 the + button is disabled 6 ms
```

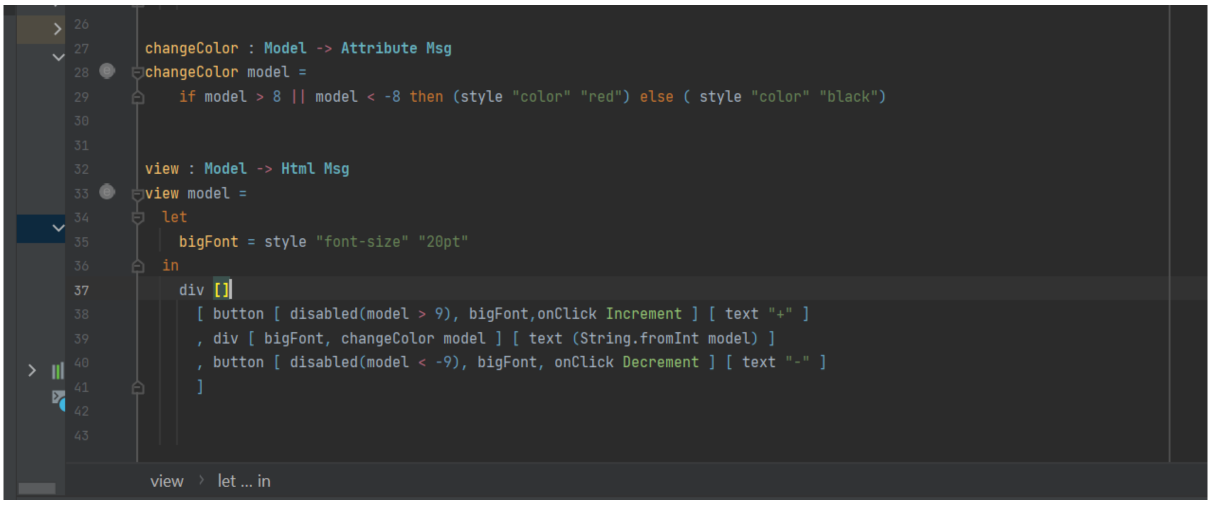
Exercise 6.2.3: Modify the Counter app to make the text red when the counter is close (is greater than 8 or less than -8) to 10 or -10.

First, I tried doing this, but the result was that the text was red all the time.



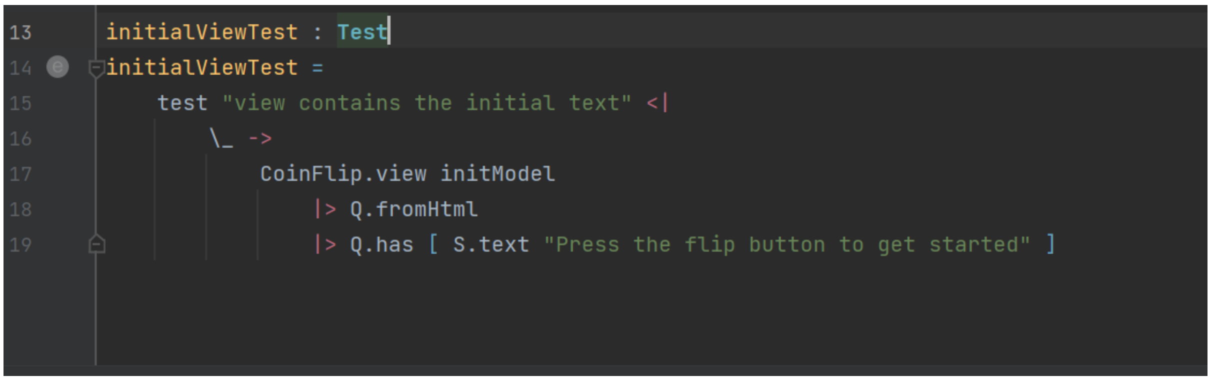
```
view : Model -> Html Msg
view model =
  let
    bigFont = style "font-size" "20pt"
  in
    div []
      [ button [ disabled(model > 9), bigFont,onClick Increment ] [ text "+" ]
      , div [ bigFont, style "color" "red" ] [ text (String.fromInt model) ]
      , button [ disabled(model < -9), bigFont, onClick Decrement ] [ text "-" ]
      ]
```

I managed to solve the problems by creating a function changeColor and then call it in the view function:



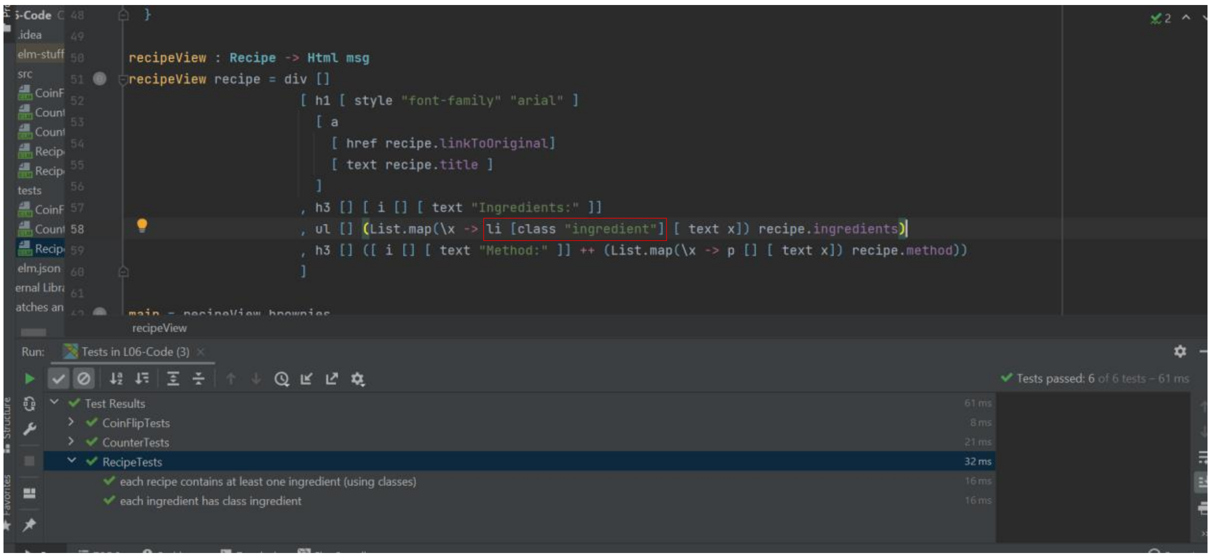
```
changeColor : Model -> Attribute Msg
changeColor model =
    if model > 8 || model < -8 then (style "color" "red") else ( style "color" "black")


view : Model -> Html Msg
view model =
  let
    bigFont = style "font-size" "20pt"
  in
    div []
      [ button [ disabled(model > 9), bigFont,onClick Increment ] [ text "+" ]
      , div [ bigFont, changeColor model ] [ text (String.fromInt model) ]
      , button [ disabled(model < -9), bigFont, onClick Decrement ] [ text "-" ]
      ]
```

Exercise 6.4.1: Write a test for the coin flip app to test that the initial view contains the text “Press the flip button to get started".



```
initialViewTest : Test
initialViewTest =
    test "view contains the initial text" <|
        \_ ->
            CoinFlip.view initModel
                |> Q.fromHtml
                |> Q.has [ S.text "Press the flip button to get started" ]
```

Exercise 6.4.2: Change your solution to Exercise 6.2.1 by adding the ingredient class to the view for each ingredient, such that the *atLeastOneIngredientClass* and *eachIngredientHasClassIngredient* both pass.



```
}

recipeView : Recipe -> Html msg
recipeView recipe = div []
                        [ h1 [ style "font-family" "arial" ]
                          [ a
                            [ href recipe.linkToOriginal]
                            [ text recipe.title ]
                          ]
                        , h3 [] [ i [] [ text "Ingredients:" ]]
                        , ul [] (List.map(\x -> li [class "ingredient"] [ text x]) recipe.ingredients)
                        , h3 [] ([ i [] [ text "Method:" ]] ++ (List.map(\x -> p [] [ text x]) recipe.method))
                        ]
```

```
Run: Tests in L06-Code (3)
Tests passed: 6 of 6 tests – 61 ms
Test Results                                                  61 ms
  CoinFlipTests                                                8 ms
  CounterTests                                                21 ms
  RecipeTests                                                 32 ms
    each recipe contains at least one ingredient (using classes)  16 ms
    each ingredient has class ingredient                      16 ms
```

Question 6.5.1: What are the 3 components of the Elm Architecture?
- Model
- View
- Update

Question 6.5.2: What is the fundamental difference between a command and a message?

A command is the task that we give to Elm to deal with actions from the outside world (generate a random thing in case of the coin flip problem). It is necessary to use commands to make more useful and interactive apps. (the recipient is the outside world like the browser)

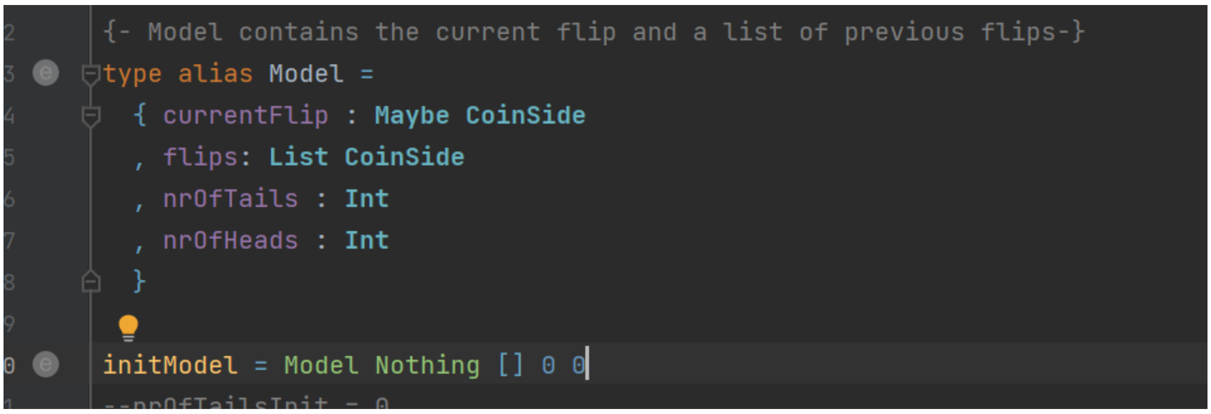
A message is what the update function receives in order to return a new model based on the initial one. (no interactions with outside world)

Question 6.5.3: What are the 2 steps of a command?

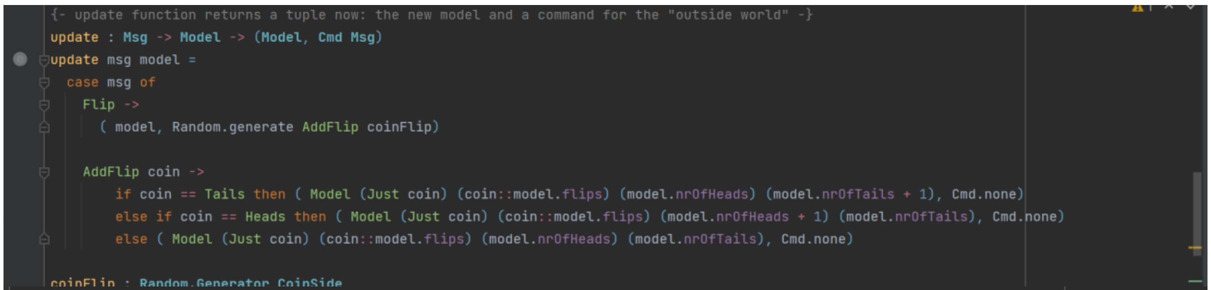
The command is given to Elm runtime in the update function and then, when Elm runtime completes the given task it sends us a message with the result.

Exercise 6.6.1: Modify the Coin flip app to display the number of heads and tails outcomes so far, in two ways:
1. Keep the number in the Model and simply display it in the view
2. Compute the values from the flips _eld of the Model each time in the view

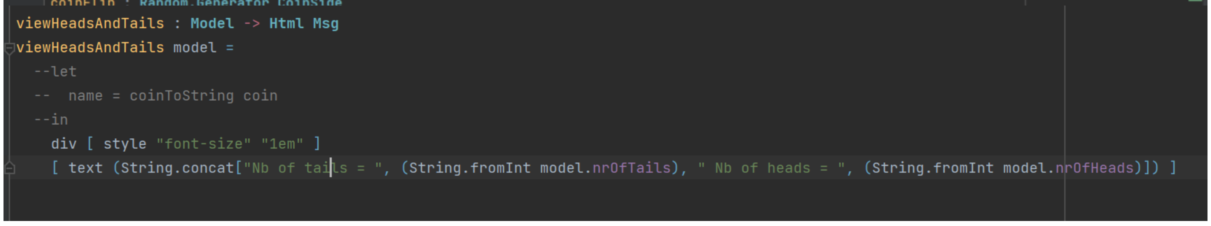
- *For the first method, I modified the following things:*



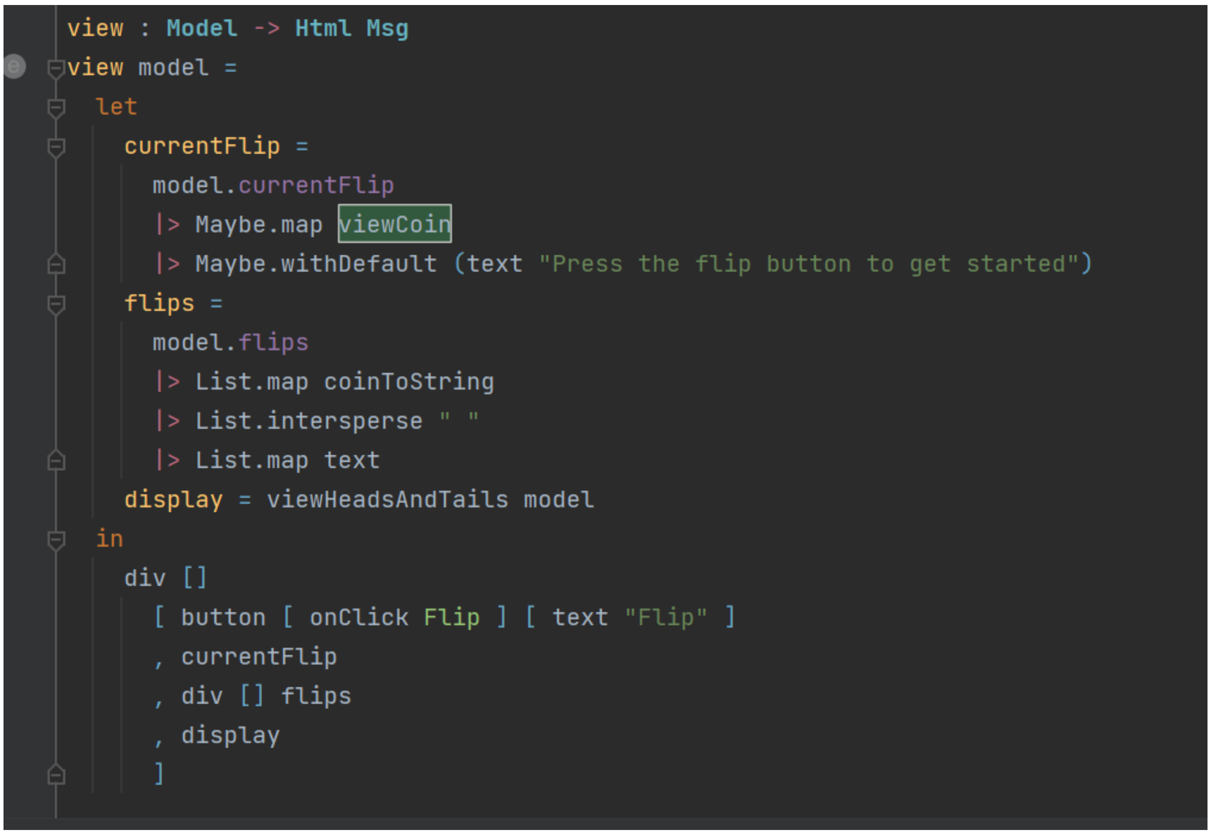
```elm
{- Model contains the current flip and a list of previous flips-}
type alias Model =
  { currentFlip : Maybe CoinSide
  , flips: List CoinSide
  , nrOfTails : Int
  , nrOfHeads : Int
  }

initModel = Model Nothing [] 0 0
--nrOfTailsInit = 0
```



```elm
{- update function returns a tuple now: the new model and a command for the "outside world" -}
update : Msg -> Model -> (Model, Cmd Msg)
update msg model =
  case msg of
    Flip ->
      ( model, Random.generate AddFlip coinFlip)

    AddFlip coin ->
        if coin == Tails then ( Model (Just coin) (coin::model.flips) (model.nrOfHeads) (model.nrOfTails + 1), Cmd.none)
        else if coin == Heads then ( Model (Just coin) (coin::model.flips) (model.nrOfHeads + 1) (model.nrOfTails), Cmd.none)
        else ( Model (Just coin) (coin::model.flips) (model.nrOfHeads) (model.nrOfTails), Cmd.none)

coinFlip : Random.Generator CoinSide
```

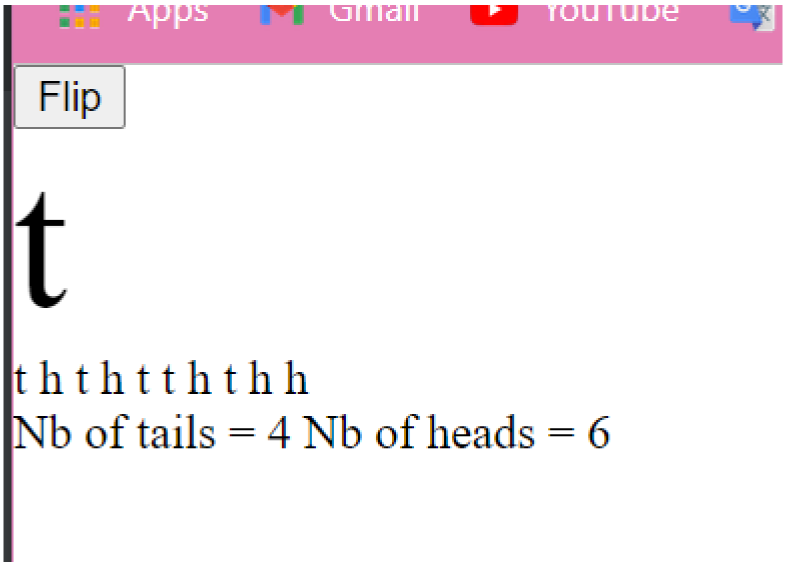


```elm
viewHeadsAndTails : Model -> Html Msg
viewHeadsAndTails model =
  --let
  --  name = coinToString coin
  --in
    div [ style "font-size" "1em" ]
    [ text (String.concat["Nb of tails = ", (String.fromInt model.nrOfTails), " Nb of heads = ", (String.fromInt model.nrOfHeads)]) ]
```



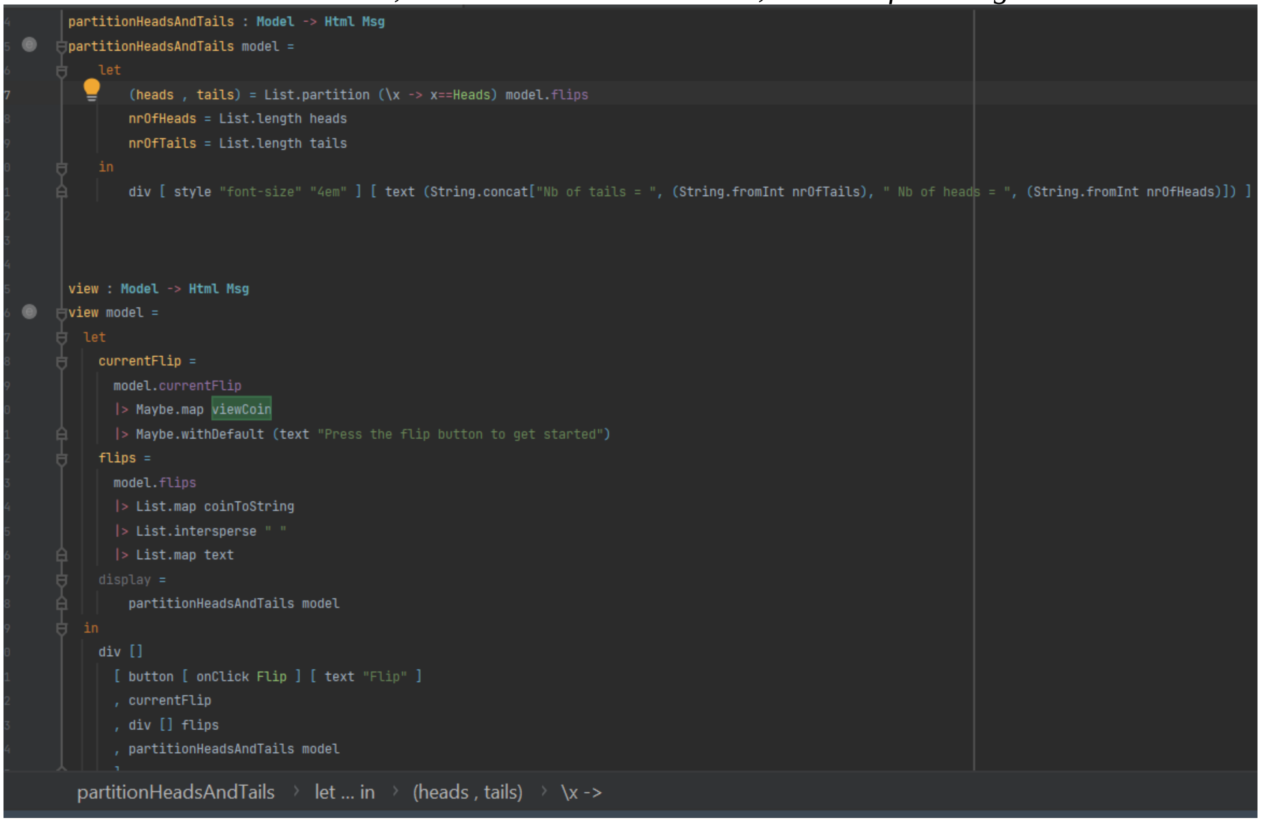
```elm
view : Model -> Html Msg
view model =
  let
    currentFlip =
      model.currentFlip
      |> Maybe.map viewCoin
      |> Maybe.withDefault (text "Press the flip button to get started")
    flips =
      model.flips
      |> List.map coinToString
      |> List.intersperse " "
      |> List.map text
    display = viewHeadsAndTails model
  in
    div []
      [ button [ onClick Flip ] [ text "Flip" ]
      , currentFlip
      , div [] flips
      , display
      ]
```

Unfortunately, it does not work how it should and I couldn't find what's wrong so far ☹

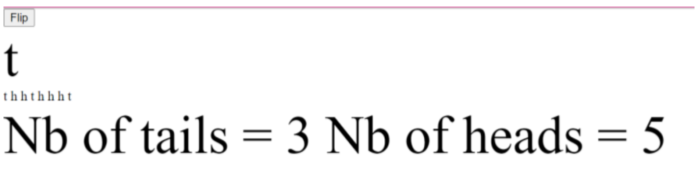


```
t

t h t h t t h t h h
Nb of tails = 4 Nb of heads = 6
```

- *For the second method, which seems to work better, I did the following:*



```elm
partitionHeadsAndTails : Model -> Html Msg
partitionHeadsAndTails model =
    let
        (heads , tails) = List.partition (\x -> x==Heads) model.flips
        nrOfHeads = List.length heads
        nrOfTails = List.length tails
    in
        div [ style "font-size" "4em" ] [ text (String.concat["Nb of tails = ", (String.fromInt nrOfTails), " Nb of heads = ", (String.fromInt nrOfHeads)]) ]



view : Model -> Html Msg
view model =
  let
    currentFlip =
      model.currentFlip
      |> Maybe.map viewCoin
      |> Maybe.withDefault (text "Press the flip button to get started")
    flips =
      model.flips
      |> List.map coinToString
      |> List.intersperse " "
      |> List.map text
    display =
        partitionHeadsAndTails model
  in
    div []
      [ button [ onClick Flip ] [ text "Flip" ]
      , currentFlip
      , div [] flips
      , partitionHeadsAndTails model
```
partitionHeadsAndTails › let ... in › (heads , tails) › \x ->



```
t
t h h t h h h t
Nb of tails = 3 Nb of heads = 5
```