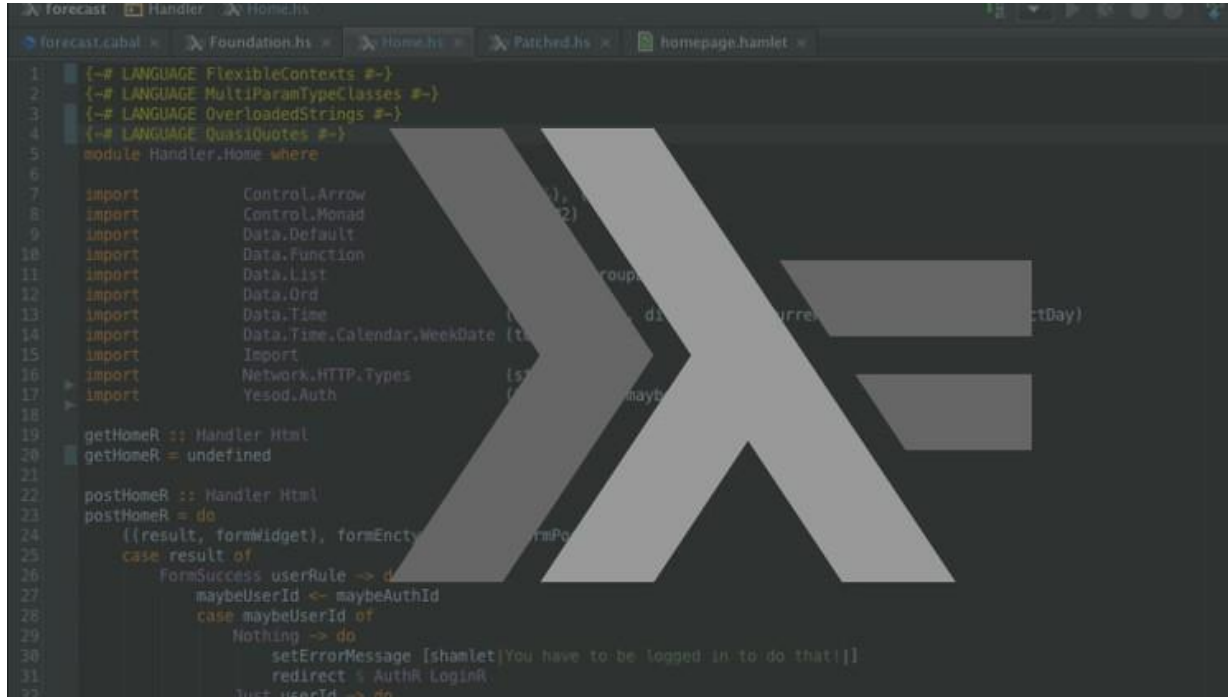


Decomposition of Tuples into Patterns



```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate
15 import Import
16 import Network.HTTP.Types
17 import Yesod.Auth
18
19 getHomeR :: Handler Html
20 getHomeR = undefined
21
22 postHomeR :: Handler Html
23 postHomeR = do
24   ((result, formWidget), formEncrypt)
25   case result of
26     FormSuccess userRule -> do
27       maybeUserId <- maybeAuthId
28       case maybeUserId of
29         Nothing -> do
30           setErrorMessage [shamlet|You have to be logged in to do that!|]
31           redirect % AuthR.LoginR
32         Just userId -> do
```

Decomposition of Tuples into Patterns

Ugly:

```
distance :: (Float, Float) -> (Float, Float) -> Float
    computes the distance between two 2D points, each given by a tuple

distance p1 p2 = sqrt ((fst p1 - fst p2)^2 + (snd p1 - snd p2)^2)
```

Better: Decompose by patterns to the parameters themselves:

```
distance (x1, y1) (x2, y2) = sqrt ((x1 - x2)^2 + (y1 - y2)^2)
```

Decomposition of Tuples into Patterns

Also: Decompose by patterns using local names:

```
distance p1 p2 = sqrt (sqr dx + sqr dy)
  where
    (x1, y1) = p1
    (x2, y2) = p2
    dx = x1 - x2
    dy = y1 - y2
    sqr x = x * x
```

Lucas Bazilio Youtube Channel

