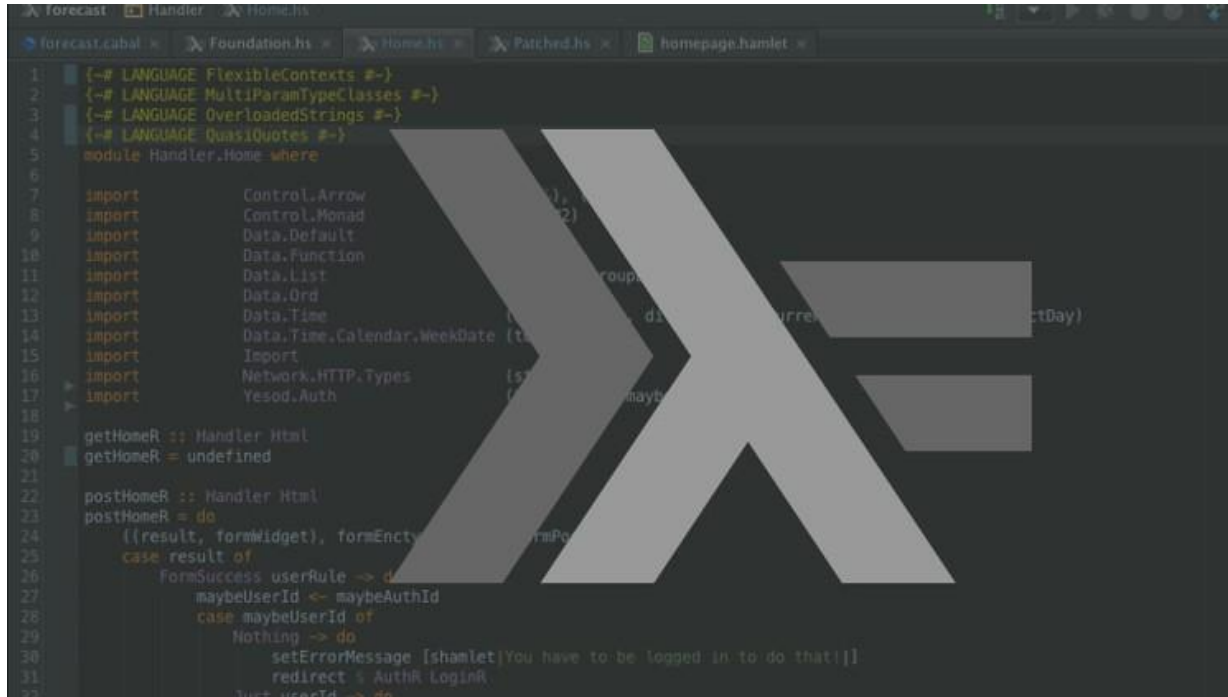


Do Notation: Example

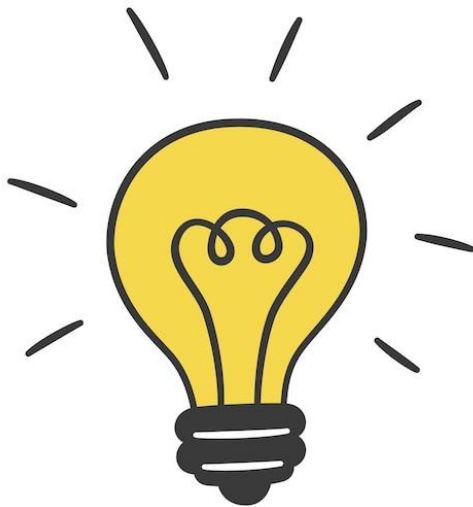


```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import 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), formEnctype, formPost)
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
```

Do Notation



The **do notation** is syntactic sugar to facilitate the use of monads.
⇒ With do, functional code *looks like* imperative code with assignments.



Do Notation



The computations can be sequenced:

```
do { c1 ; c2 }
```

≡

```
do  
  c1  
  c2
```

≡

```
c1 >> c2
```

≡

```
c1 >>= \_ -> c2
```

And with <- extract its results:

```
do { x <- c1 ; c2 }
```

≡

```
do  
  x <- c1  
  c2
```

≡

```
c1 >>= \x -> c2
```

Do Notation: Example



We have associative lists with information about car owners, their license plates, their models and their emission labels:

```
data Model = Clio | Audi | Cadillac | Nissan deriving (Eq, Show)
data Label = B | C | Eco | Zero deriving (Eq, Show)

registrations = [("Albert", 3526), ("Peter", 8427), ("Sofia", 7383), ("Olivia", 5913)]
models = [(3526, Audi), (8427, Clio), (7383, Nissan), (5913, Cadillac)]
labels = [(Clio, Zero), (Audi, C), (Cadillac, B), (Nissan, Eco)]
```

Do Notation: Example



Given an owner name, we want to know what their emissions label is:

```
label :: String -> Maybe Label
```

It's Maybe because, maybe the owner doesn't exist, or we don't have his registration, or we don't have his model, or we don't have his label...

A useful predefined function to use:

```
lookup :: Eq a => a -> [(a, b)] -> Maybe b
```

Applying Do Notation



```
label name = do  
  reg <- lookup name registrations  
  mod <- lookup reg models  
  lookup mod labels
```

Transformation from **do notation** to functional:

```
label name =  
  lookup name registrations >=> \mat ->  
  lookup reg models >=> \mod ->  
  lookup mod labels
```

Instructor Youtube Channel: Lucas Science

