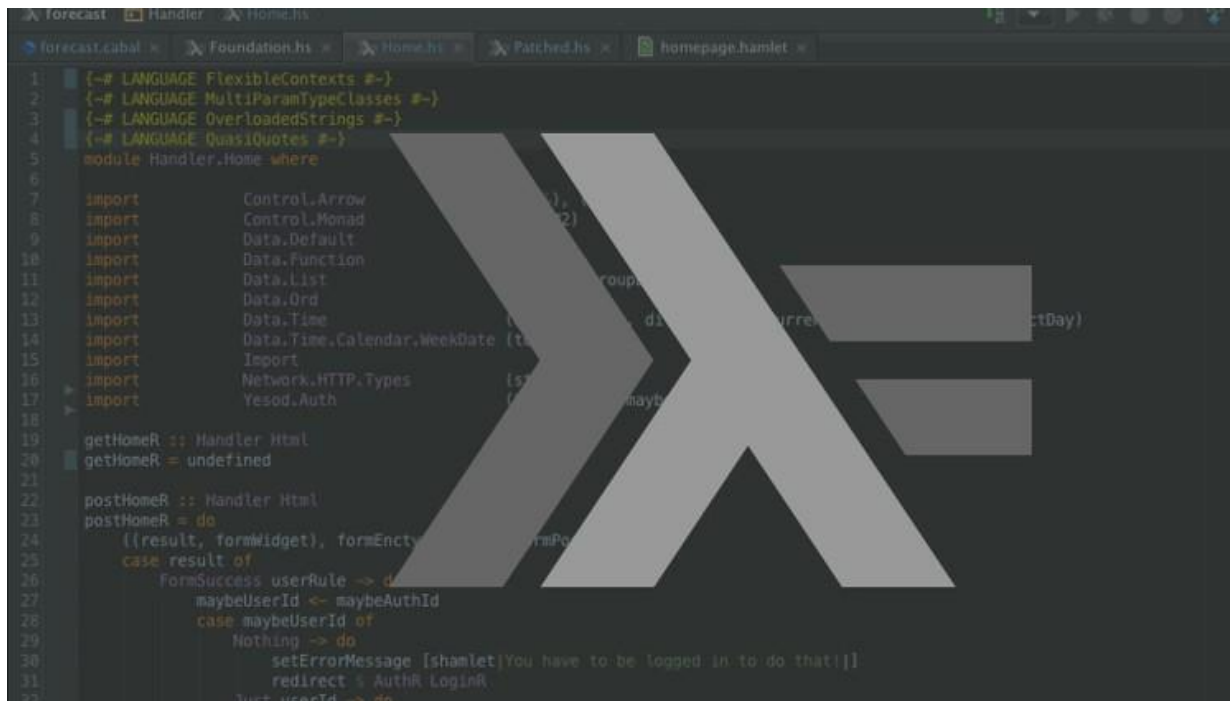


# Definition with Patterns



```
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), formEnctype) <- runFormPost
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
```

# Definition with Patterns

Functions can be defined with patterns:

```
factorial :: Integer -> Integer
  -- calculates the factorial of a natural

factorial 0 = 1
factorial n = n * factorial (n - 1)
```

The evaluation of the patterns is from top to bottom and returns the result of the first matching branch.

Patterns are considered more elegant than the `if-then-else` and they have many more applications.

# Definition with Patterns

`_` represents an **anonymous variable**: (there is no relation between different `_`)

```
nand :: Bool -> Bool -> Bool           -- negated conjunction
nand True True = False
nand _ _ = True
```