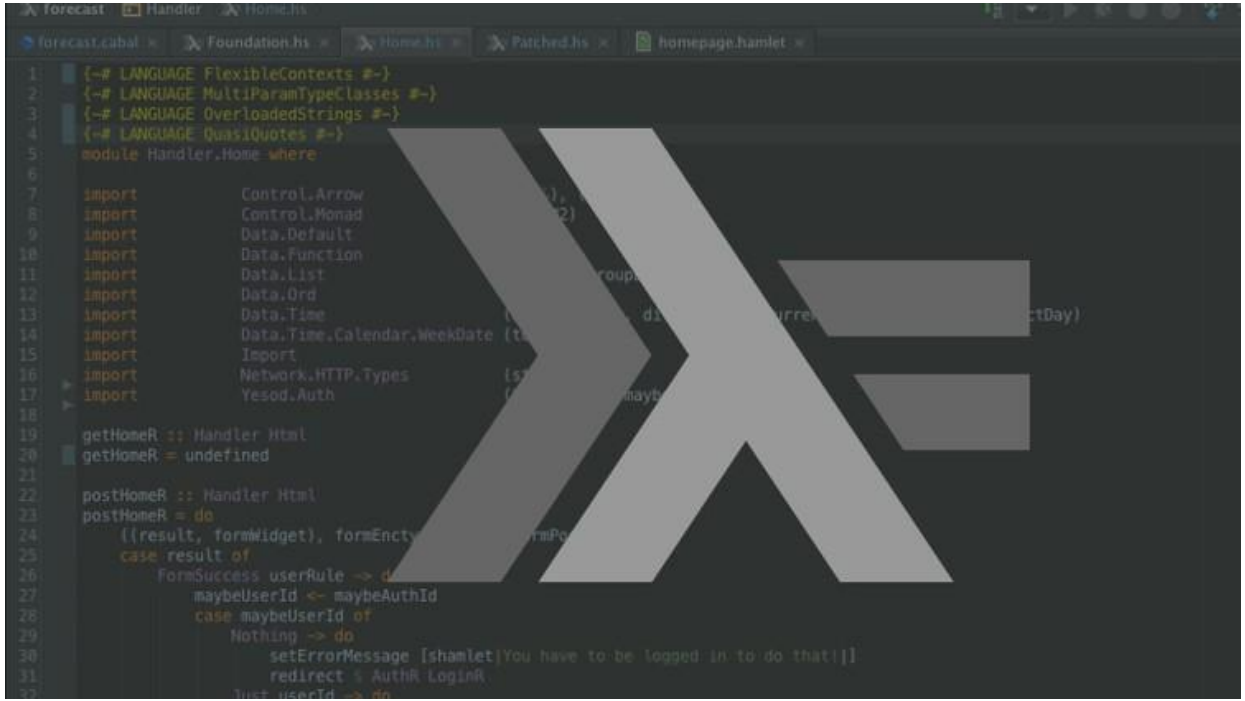
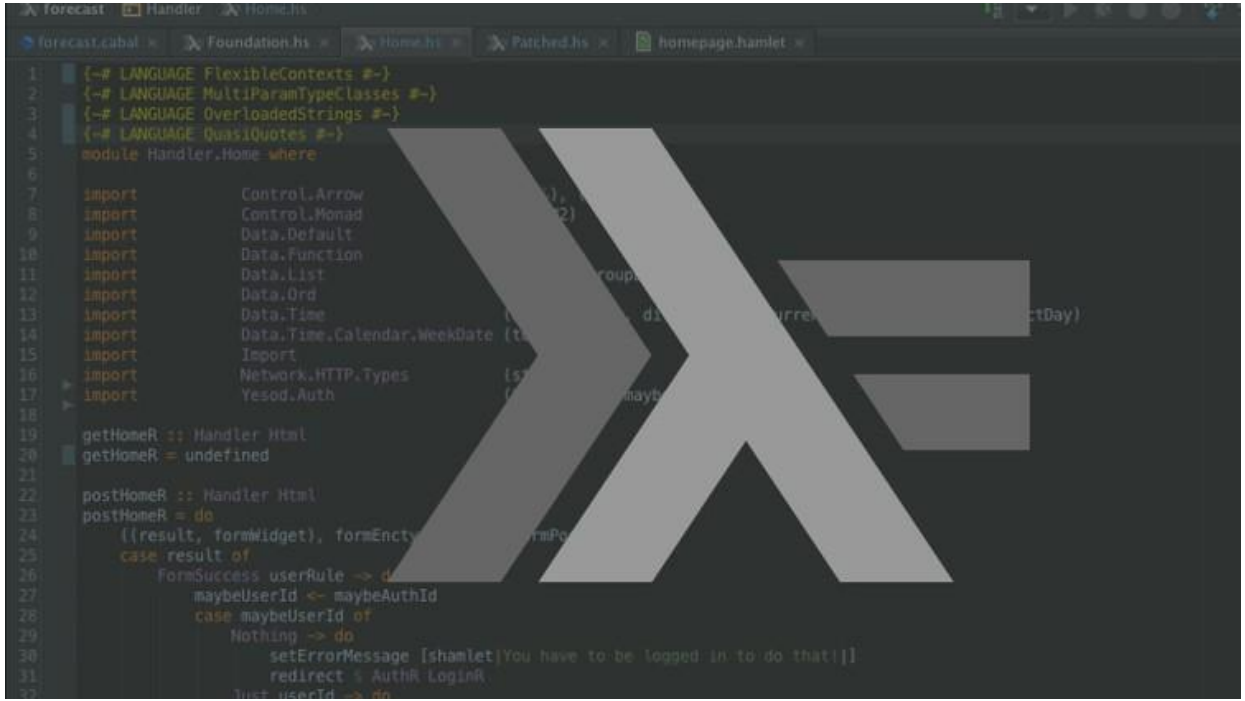
Monads


forecast.cabal
Foundation.hs
Home.hs
Patched.hs
homepage.hamlet
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where
import Control.Arrow
import Control.Monad
import Data.Default
import Data.Function
import Data.List
import Data.Ord
import Data.Time
import Data.Time.Calendar.WeekDate
import Import
import Network.HTTP.Types
import Yesod.Auth
getHomeR :: Handler Html
getHomeR = undefined
postHomeR :: Handler Html
postHomeR = do
((result, formWidget), formEncty
case result of
FormSuccess userRule -> d
maybeUserId <- maybeAuthId
case maybeUserId of
Nothing -> do
setErrorMessage [shamlet|You have to be logged in to do that!|]
redirect $ AuthR LoginR

Introduction to Monads



Lucas Bazilio - Udemy

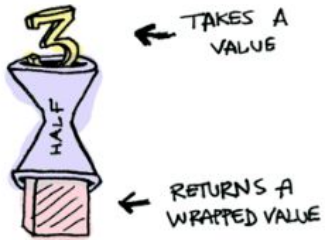
Monads

Consider that ***half*** is a function that only makes sense on even numbers:

```haskell
half :: Int -> Maybe Int

half x
    | even x    = Just (div x 2)
    | otherwise = Nothing
```

We can see the function like this: Given a value, return a packed value.



But then we can't stuff packed values into it!

We need a function that unpacks, applies `half` and leaves encapsulated.

This function is called `>>=` (pronounced *bind*)

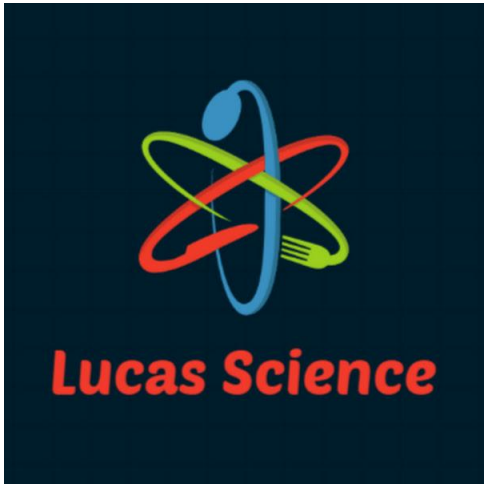
```
λ> Just 40 >>= half    👉 Just 20
λ> Just 31 >>= half    👉 Nothing
λ> Nothing >>= half    👉 Nothing

λ> Just 20 >>= half >>= half              👉 Just 5
λ> Just 20 >>= half >>= half >>= half   👉 Nothing
```

The operator `>>=` is an operation of the class `Monad`:

```
class Applicative m => Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    -- i més coses
```

The type `Maybe` is istance of `Monad`:

```
instance Monad Maybe where
    Nothing >>= f   =   Nothing
    Just x  >>= f   =   f x
```

Instructor Youtube Channel: Lucas Science