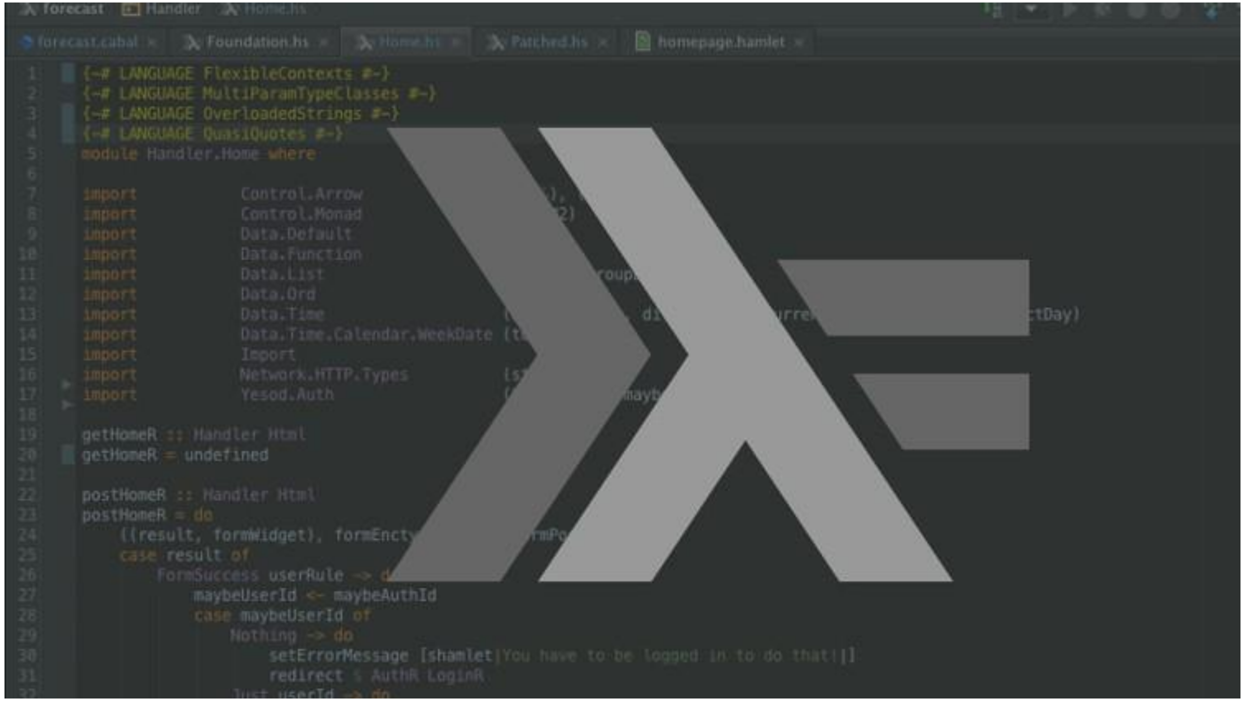
Monads



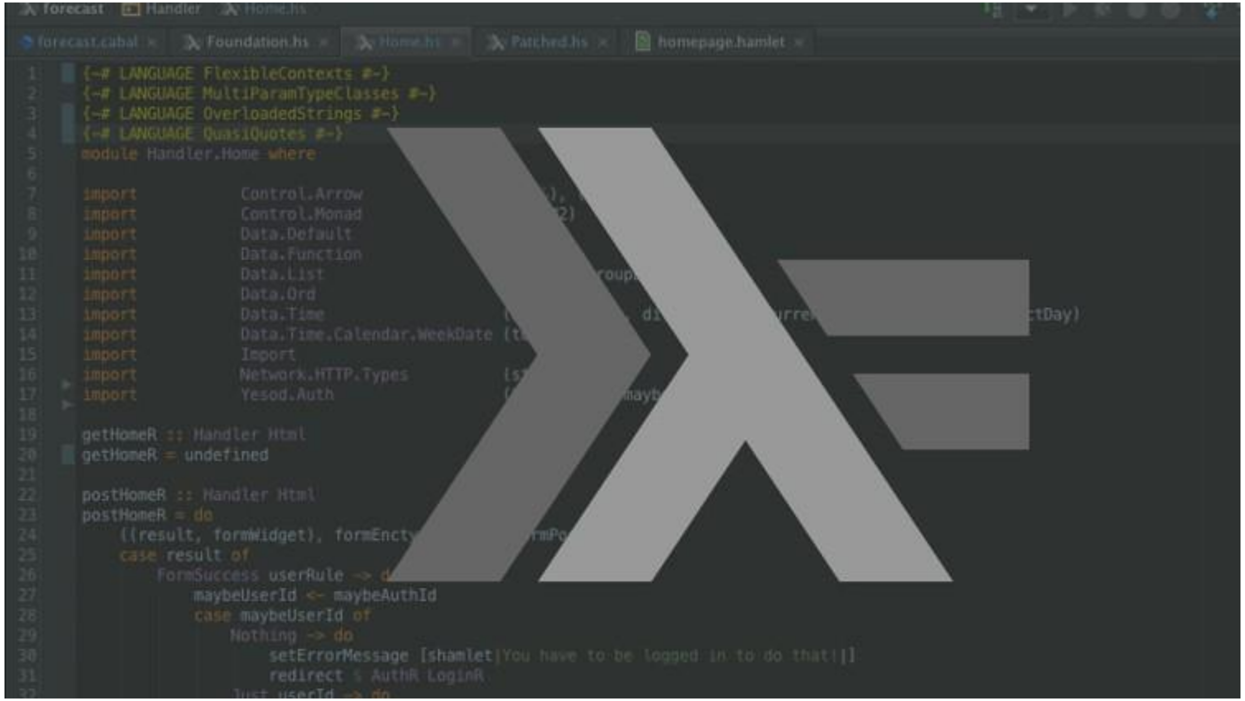
```
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where

import           Control.Arrow                 ),
import           Control.Monad                 2)
import           Data.Default
import           Data.Function
import           Data.List                       roup
import           Data.Ord
import           Data.Time                 (       , di      urre              tDay)
import           Data.Time.Calendar.WeekDate (t
import           Import
import           Network.HTTP.Types          (s
import           Yesod.Auth                  (       mayb

getHomeR :: Handler Html
getHomeR = undefined

postHomeR :: Handler Html
postHomeR = do
    ((result, formWidget), formEncty          rmPo
    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
                Just userId -> do
```

Monads Instances



```
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where

import           Control.Arrow                    ), (
import           Control.Monad                    2)
import           Data.Default
import           Data.Function
import           Data.List                          roup
import           Data.Ord
import           Data.Time                (       , di      urre          ctDay)
import           Data.Time.Calendar.WeekDate (t
import           Import
import           Network.HTTP.Types          (s
import           Yesod.Auth                  (          mayb

getHomeR :: Handler Html
getHomeR = undefined

postHomeR :: Handler Html
postHomeR = do
    ((result, formWidget), formEncty          rmPo
    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
                Just userId -> do
```

Monads Operations

In fact, monads have three operations:

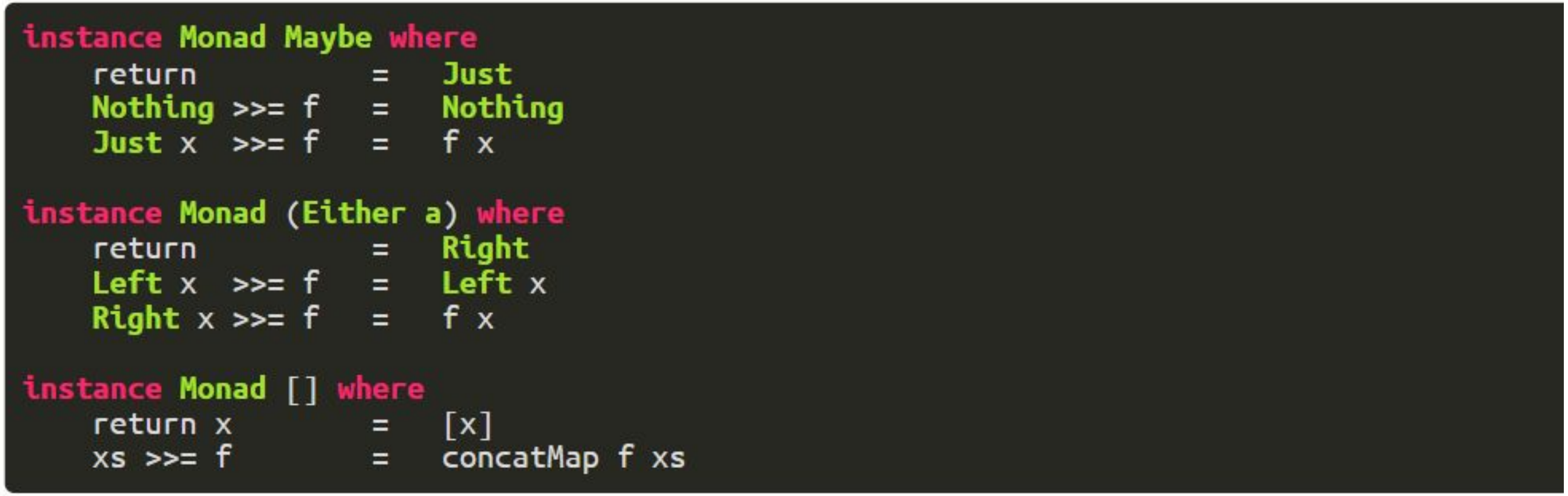
```haskell
class Monad m where
    return :: a -> m a
    (>>=)  :: m a -> (a -> m b) -> m b
    (>>)   :: m a -> m b -> m b

    r >> k   =   r >>= (\_ -> k)
```

- `return` wrap.
- `>>=` unwrap, apply and wrap.
- `>>` is purely esthetic.

Monads Instances

The types `Maybe`, `Either a` and `[]` are instances of `Monad`:



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

instance Monad (Either a) where
    return            =   Right
    Left x  >>= f     =   Left x
    Right x >>= f     =   f x

instance Monad [] where
    return x          =   [x]
    xs >>= f          =   concatMap f xs
```

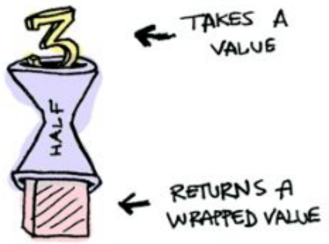
Monads

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
```

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

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

Instructor Youtube Channel: Lucas Science

Lucas Science

SUBSCRIBE