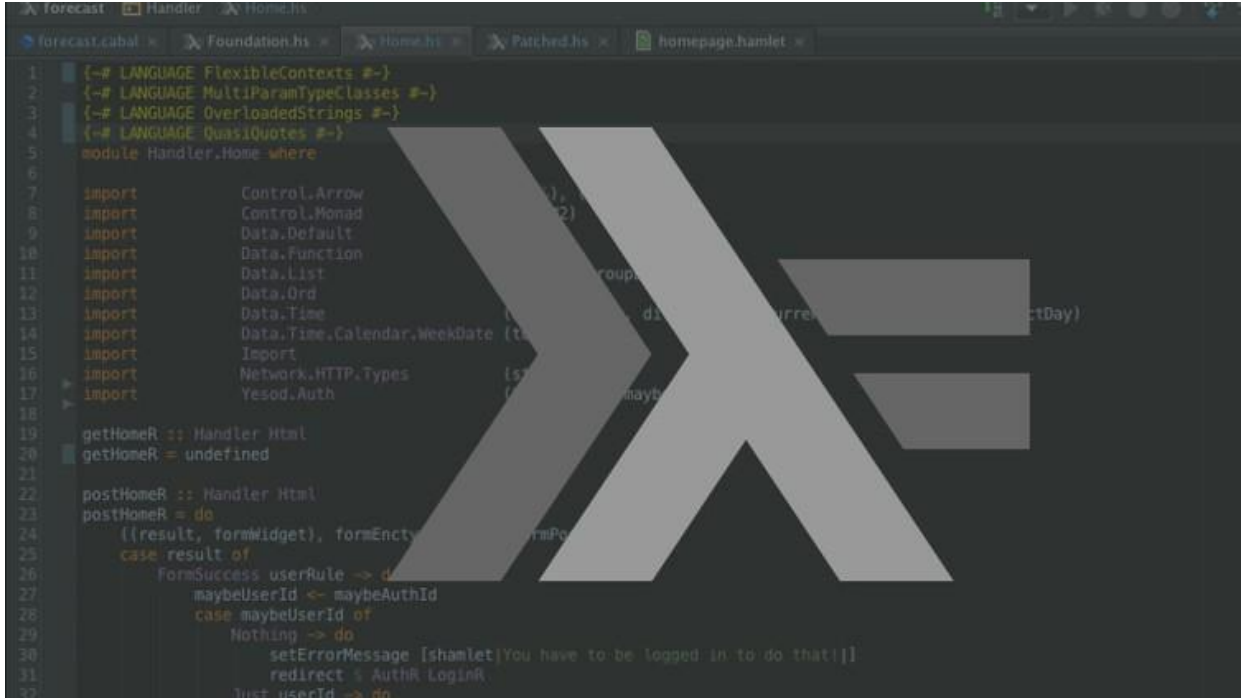
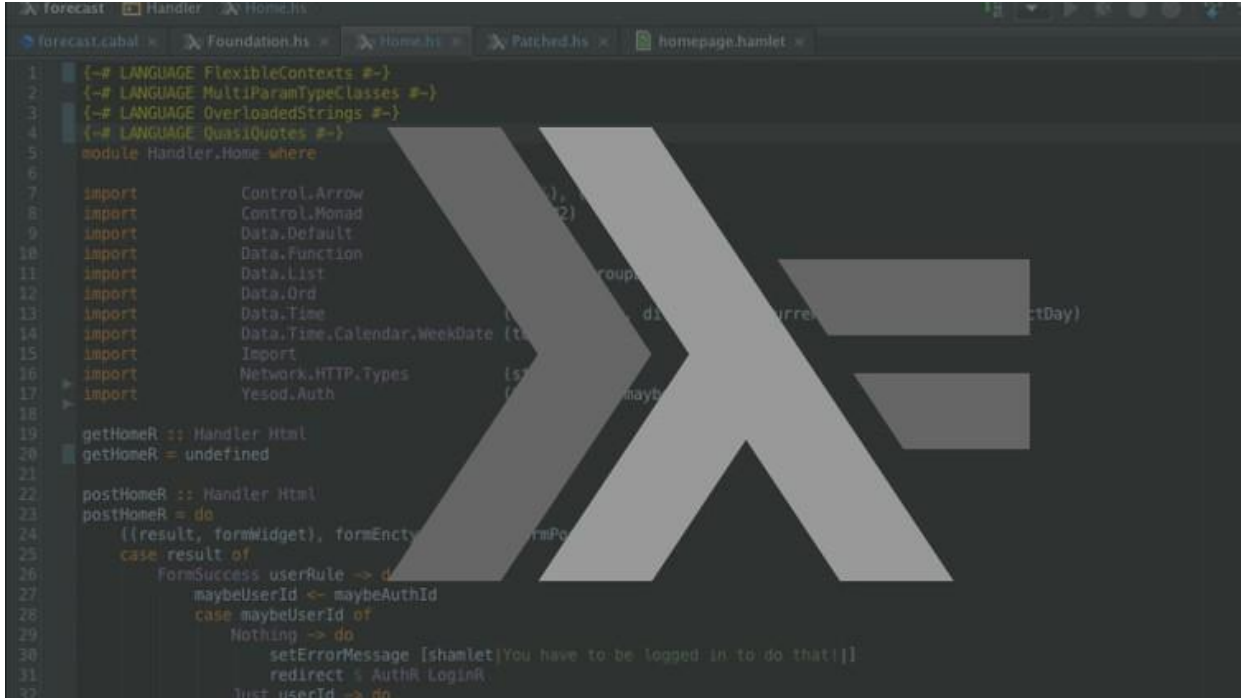
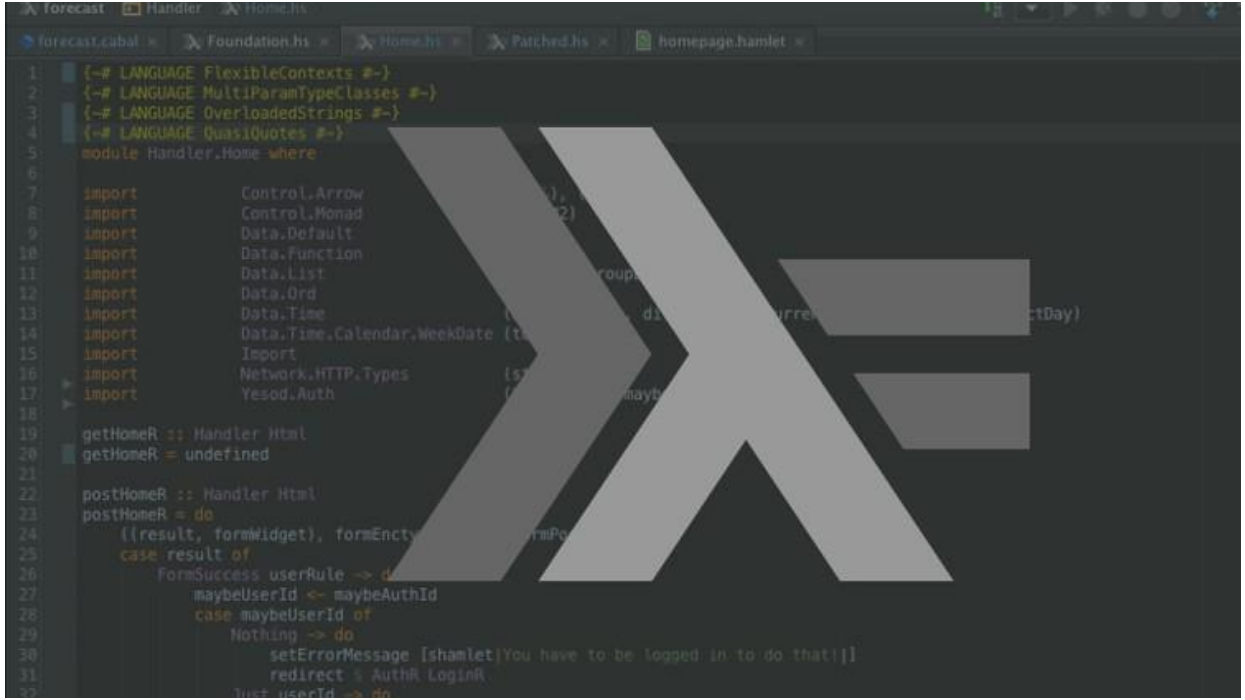
Final Exams


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
case result of
FormSuccess userRule
maybeUserId <- maybeAuthId
case maybeUserId of
Nothing -> do
setErrorMessage [shamlet|You have to be logged in to do that!|]
redirect $ AuthR LoginR

Exam 1


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
case result of
FormSuccess userRule
maybeUserId <- maybeAuthId
case maybeUserId of
Nothing -> do
setErrorMessage [shamlet|You have to be logged in to do that!|]
redirect % AuthR LoginR

Problem 2


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
case result of
FormSuccess userRule
maybeUserId <- maybeAuthId
case maybeUserId of
Nothing -> do
setErrorMessage [shamlet|You have to be logged in to do that!|]
redirect $ AuthR LoginR

Problem 2

Define a `Rational` type to manipulate positive rational numbers with operations by:

1. Construct a rational through a natural numerator and denominator.
2. Obtain the numerator of its simplified form.
3. Obtain the denominator of its simplified form.

Also, make Rational a member of class Eq and class Show, making rationals display in the form "`x/y`".

Problem 2

Follow this interface:

```
data Rational = ...
rational :: Integer -> Integer -> Rational
numerator :: Rational -> Integer
denominator :: Rational -> Integer
```

If you want, you can use the standard gcd function which returns the greatest common divisor of two naturals.

Problem 2

Input	Output
`numerator (rational 1 2)`	`-> 1`
`denominator (rational 1 2)`	`-> 2`
`numerator (rational 2 4)`	`-> 1`
`denominator (rational 2 4)`	`-> 2`
`rational 1 2`	`-> 1/2`
`rational 2 4`	`-> 1/2`
`rational 1 2 == rational 2 4`	`-> True`
`rational 1 2 == rational 1 3`	`-> False`

Problem 2

EXECUTION OF THE PROGRAM:

Change at the end the name of the type `Rational` to be `Rationnal` so Haskell does not get confused with the predefined type.

Instructor Youtube Channel: Lucas Science

Lucas Science

SUBSCRIBE