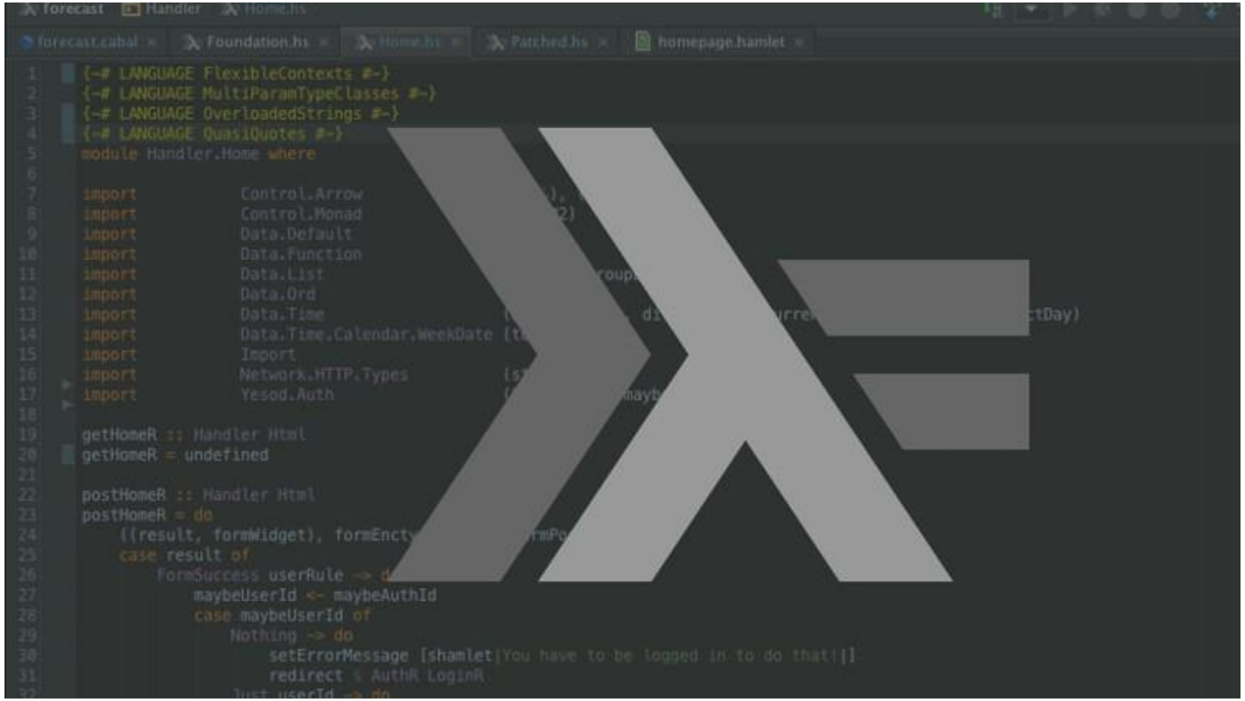
Instantiations of Applicative



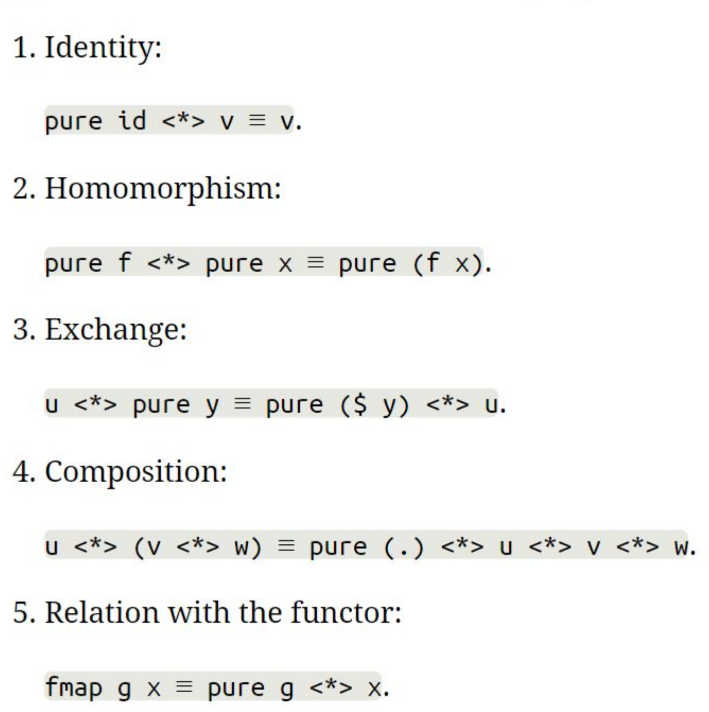
```
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE QuasiQuotes #-}
module Handler.Home where

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
```

Laws of Applicatives



1. Identity:

```
pure id <*> v ≡ v.
```

2. Homomorphism:

```
pure f <*> pure x ≡ pure (f x).
```

3. Exchange:

```
u <*> pure y ≡ pure ($ y) <*> u.
```

4. Composition:

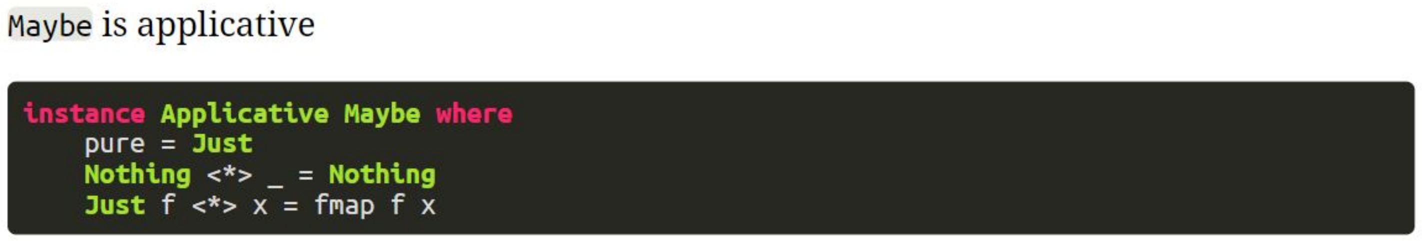
```
u <*> (v <*> w) ≡ pure (.) <*> u <*> v <*> w.
```

5. Relation with the functor:

```
fmap g x ≡ pure g <*> x.
```

Lucas Bazilio - Udemy

Instantiations of Applicatives



`Maybe` is applicative

```haskell
instance Applicative Maybe where
    pure = Just
    Nothing <*> _ = Nothing
    Just f <*> x = fmap f x
```

Instantiations of Applicatives

`Maybe` is applicative

```haskell
instance Applicative Maybe where
    pure = Just
    Nothing <*> _ = Nothing
    Just f <*> x = fmap f x
```

`Either a` is applicative

```haskell
instance Applicative (Either a) where
    pure = Right
    Left x <*> _ = Left x
    Right f <*> x = fmap f x
```

Instantiations of Applicatives

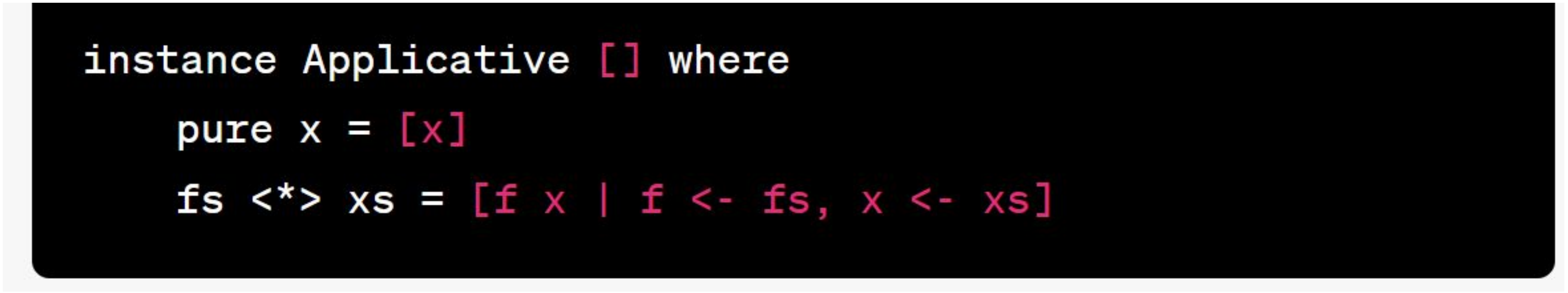
`Maybe` is applicative

```haskell
instance Applicative Maybe where
    pure = Just
    Nothing <*> _ = Nothing
    Just f <*> x = fmap f x
```

`Either a` is applicative

```haskell
instance Applicative (Either a) where
    pure = Right
    Left x <*> _ = Left x
    Right f <*> x = fmap f x
```

Instantiations of Applicatives

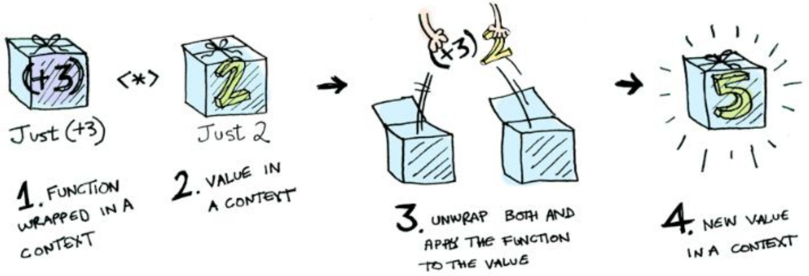


```haskell
instance Applicative [] where
    pure x = [x]
    fs <*> xs = [f x | f <- fs, x <- xs]
```

Summary

Applicatives allow functions within a container to be applied to objects within the same container.

- `pure` constructs a container with a value.
- `<*>` applies a function inside a container to values inside a container:


(+3)
<*>
2
Just (+3)
Just 2
1. FUNCTION WRAPPED IN A CONTEXT
2. VALUE IN A CONTEXT
(+3) 2
3. UNWRAP BOTH AND APPLY THE FUNCTION TO THE VALUE
5
4. NEW VALUE IN A CONTEXT