Monads and Typeclasses

Inspired by the book
“Finding Success and Failure in Haskell”
by Julie Moronuki and Chris Martin

Rushali Sarkar

August 2021

Introduction

Introduction

Monad is a typeclass and a typeclass defines a set of generic functions that work with a set of types.

```
Prelude> :type (+)
(+) :: Num a => a -> a -> a
```

Why Monads?

Why Monads ?

Monad encapsulates the idea of merging two things together into a single one, with the possibility that one of those things is irrelevant.

Inspired from the Article by Elviro Rocca
https://broomburgo.github.io/fun-ios/post/why-monads/

Let’s Get Coding

Code

Code with Error

```haskell
errorHead :: [a] -> a
errorHead (x: xs) = x
```

A Possible Fix

```haskell
safeHead :: [a] -> Maybe a
safeHead [] = Nothing
safeHead (x: xs) = Just x
```

Another Example

Validate Password

```haskell
validatePassword :: String -> Maybe String
validatePassword password =
  case (cleanWhiteSpace password) of
    Nothing -> Nothing
    Just password2 ->
      case (requireAlphaNum password2) of
        Nothing -> Nothing
        Just password3 ->
          case (checkPasswordLength password3) of
            Nothing -> Nothing
            Just password4 -> password4
```

Another Example

Sample Code

```
convertString x = map toUpper (reverse x)
```

Using Infix Function Composition Operator

```
convertString = map toUpper.reverse
```

Bind Operator

```
 Prelude> :type (»=)
(»=) :: Monad m => m a -> (a -> m b) -> m b
```

Monads making life Easy

```
validatePassword :: String -> Maybe String
validatePassword password = cleanWhiteSpace password »=
requireAlphaNum »= checkPasswordLength
```

Thank You Everyone ☺