Table of Contents

Monads

Benson Joeris
http://bjoeris.com

pluralsight
hardcore developer training

Overview

- Familiar examples of monads
- Common functionality for all Monads
- Monad type class
- `do`-notation

Table of Contents

Monad Examples

- IO

```
return :: a -> IO a
```

```
unreturn :: IO a -> a
```

```
bindIO :: IO a -> (a -> IO b) -> IO b
```

Monad Examples

- IO
- List

```
singleton :: a -> [a]
```

```
unsingleton :: [a] -> a
```

```
flatMap :: [a] -> (a -> [b]) -> [b]
```

GHCi> `flatMap [1,7,11] (\x -> [x,x+1])`

Result: `[1,2,7,8,11,12]`

Monad Examples

- IO
- List
- Maybe

```
data Maybe a = Nothing | Just a
```

```
Just :: a -> Maybe a
```

```
unJust :: Maybe a -> a
```

```
bindMaybe :: Maybe a -> (a -> Maybe b) -> Maybe b
```

Monad Examples

- IO
- List
- Maybe

```
bindMaybe :: Maybe a -> (a -> Maybe b) -> Maybe b
```

GHCi>
```
bindMaybe Nothing (\x ->
    if (x==0)
    then Nothing
    else Just (2*x))
```
Result:
```
Nothing
```

Monad Examples

- IO
- List
- Maybe

```
bindMaybe :: Maybe a -> (a -> Maybe b) -> Maybe b
```

GHCi>
```
bindMaybe (Just 0) (\x ->
      if (x==0)
      then Nothing
      else Just (2*x))
```
Result:
```
Nothing
```

Monad Examples

- IO
- List
- Maybe

```
bindMaybe :: Maybe a -> (a -> Maybe b) -> Maybe b
```

```
GHCi> bindMaybe (Just 1) (\x ->
        if (x==0)
        then Nothing
        else Just (2*x))
Result: Just 2
```

Monad Examples

```
return    :: a -> IO a
singleton :: a -> [a]
just      :: a -> Maybe a
```

```
bindIO    :: IO a     -> (a -> IO b)     -> IO b
flatMap   :: [a]      -> (a -> [b])      -> [b]
bindMaybe :: Maybe a -> (a -> Maybe b) -> Maybe b
```

Table of Contents

Common Functionality for All Monads

```
return :: a -> IO a
return :: a -> [a]
return :: a -> Maybe a
```

```
bind   :: IO a     -> (a -> IO b)     -> IO b
bind   :: [a]      -> (a -> [b])      -> [b]
bind   :: Maybe a -> (a -> Maybe b) -> Maybe b
```

Common Functionality for All Monads

```
join :: IO (IO a) -> IO a
join :: [[a]] -> [a]
join :: Maybe (Maybe a) -> Maybe a
join mmx = bind mmx id
```

```
GHCi> join [[1,2,3],[4,5,6]]
Result: [1,2,3,4,5,6]
```

Common Functionality for All Monads

```
join :: IO (IO a) -> IO a
join :: [[a]] -> [a]
join :: Maybe (Maybe a) -> Maybe a
join mmx = bind mmx id
```

```
GHCi> join (Just (Just 7))
Result: Just 7
```

Common Functionality for All Monads

```
join :: IO (IO a) -> IO a
join :: [[a]] -> [a]
join :: Maybe (Maybe a) -> Maybe a
join mmx = bind mmx id
```

```
GHCi> join (Just Nothing)
Result: Nothing
```

Common Functionality for All Monads

```
join :: IO (IO a) -> IO a
join :: [[a]] -> [a]
join :: Maybe (Maybe a) -> Maybe a
join mmx = bind mmx id
```

GHCi> `join Nothing`

Result: `Nothing`

Common Functionality for All Monads

```
join :: IO (IO a) -> IO a
join :: [[a]] -> [a]
join :: Maybe (Maybe a) -> Maybe a
join mmx = bind mmx id
```

GHCi> `join Nothing`

Result: `Nothing`

Table of Contents

Monad Type Class

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

- Captures common pattern in `IO`, list, and `Maybe`

```
join :: Monad m => m (m a) -> m a
join mmx = mmx >>= id
```

- Type class of parameterized types
- Monad laws

Table of Contents

do-Notation

```
addM :: Monad m => m Int -> m Int -> m Int
addM mx my =
  mx >>= (\x -> my >>= (\y -> return (x + y)))
```

```
addM' :: Monad m => m Int -> m Int -> m Int
addM' mx m y = do
  x <- mx
  y <- my
  return (x + y)
```

do-Notation

- Syntactic sugar
- *Not* imperative code

```
do
  x <- mx
  ...
```

```
mx >>=  (\x -> ... )
```

do-Notation

```
people = ["Alice","Bob","Eve"]
items  = ["car","puppy"]
missing = do
  person <- people
  item   <- items
  return (person ++ " lost a " ++ item)
```

GHCi> `missing`

Result:

```
[ "Alice lost a car"
, "Alice lost a puppy"
, "Bob lost a car"
, "Bob lost a puppy"
, "Eve lost a car"
, "Eve lost a puppy" ]
```

Table of Contents

Summary

- Familiar monads
- Common pattern
- Monad type class
- `do`-notation