Table of Contents

Monadic Functions

Benson Joeris
http://bjoeris.com

Overview

- Monadic flow control
- Turn an ordinary function into a monadic function
- Lists and monads

Table of Contents

Control.Monad

```
import Control.Monad
```

Monadic Flow Control

- Analogous with imperative flow control constructs
- Actually just ordinary functions

Monadic Flow Control: `forM`

```
forM  :: Monad m => [a] -> (a -> m b) -> m [b]
```

```
forM_ :: Monad m => [a] -> (a -> m b) -> m ()
```

```
forM list $ do
  x <- first_action
  second_action
  ...
```

Monadic Flow Control: `forM`

```
GHCi> forM [1,2,3] print
Result: 1
Result: 2
Result: 3
```

Monadic Flow Control: `replicateM`

```
replicateM  :: Monad m => Int -> m a -> m [a]
```

```
replicateM_ :: Monad m => Int -> m a -> m ()
```

Monadic Flow Control: `forM`

```
GHCi> replicateM 3 (putStrLn "hello")
Result: hello
Result: hello
Result: hello
```

Monadic Flow Control: `when`

```
when :: Monad m => Bool -> m () -> m ()
```

```
when debug (putStrLn "Debugging")
```

Monadic Flow Control

```
forM          :: Monad m => [a] -> (a -> m b) -> m [b]
```

```
replicateM :: Monad m => Int -> m a -> m [a]
```

```
when          :: Monad m => Bool -> m () -> m ()
```

Table of Contents

Lifting: `liftM`

```
liftM :: Monad m => (a -> b) -> (m a -> m b)
```

Lifting: `liftM`

GHCi>
```
liftM (1+) (Just 3)
```
Result:
```
Just 4
```

Lifting: `liftM2`

```
liftM2 :: Monad m =>
          (a1 -> a2 -> b) -> m a1 -> m a2 -> m b
```

```
liftM3 :: Monad m =>
          (a1 -> a2 -> a3 -> b) ->
          m a1 -> m a2 -> m a3 -> m b
```

```
liftM4 :: Monad m =>
          (a1 -> a2 -> a3 -> a4 -> b) ->
          m a1 -> m a2 -> m a3 -> m a4 -> m b
```

```
liftM5 :: Monad m =>
          (a1 -> a2 -> a3 -> a4 -> a5 -> b) ->
        m a1 -> m a2 -> m a3 -> m a4 -> m a5 -> m b
```

Lifting: `liftM2`

GHCi>
```
liftM2 (+) (Just 3) (Just 5)
```
Result:
```
Just 8
```

Table of Contents

Monadic List Functions: `mapM`

```
mapM :: Monad m => (a -> m b) -> [a] -> m [b]
```

```
forM :: Monad m => [a] -> (a -> m b) -> m [b]
```

Monadic List Functions: `mapM`

```
GHCi> mapM print [1,2,3]
Result: 1
Result: 2
Result: 3
```

Monadic List Functions: `filterM`

```
filterM :: Monad m => (a -> m Bool) -> [a] -> m [a]
```

Monadic List Functions: `filterM`

```
askToKeep :: Int -> IO Bool
askToKeep x = do
  putStrLn ("keep " ++ (show x) ++ "?")
  (c : _) <- getLine
  return (c == 'y')

askWhichToKeep :: [Int] -> IO [Int]
askWhichToKeep xs =
  filterM askToKeep xs
```

Monadic List Functions: `foldM`

```
foldM :: Monad m => (a -> b -> m a) ->
                    a -> [b] -> m a
```

Monadic List Functions: `foldM`

```
sayAddition :: Int -> Int -> IO Int
sayAddition x y = do
  let z = x + y
  putStrLn ((show x) ++ " + " ++
            (show y) ++ " = " ++
            (show z))
  return z

talkingSum :: [Int] -> IO Int
talkingSum xs = foldM sayAddition 0 xs
```

Monadic List Functions: `foldM`

```
GHCi> talkingSum [1,2,3]
Result: 0 + 1 = 1
Result: 1 + 2 = 3
Result: 3 + 3 = 6
Result: 6
```

Monadic List Functions

```
mapM     :: Monad m => (a -> m b) -> [a] -> m [b]
```

```
filterM :: Monad m => (a -> m Bool) -> [a] -> m [a]
```

```
foldM    :: Monad m => (a -> b -> m a) ->
                       a -> [b] -> m a
```

Table of Contents

Summary

- Monadic Flow Control
  - `forM`
  - `replicateM`
  - `when`
- Lifting
  - `liftM`
  - `liftM2,...`
- Monadic List Functions
  - `mapM`
  - `filterM`
  - `foldM`
- More: `Control.Monad` documentation