Monad (1A)

Young Won Lim
6/6/17

Based on

Haskell in 5 steps
https://wiki.haskell.org/Haskell_in_5_steps

Generator

let removeLower x=[c| c<-x, c `elem` ['A'..'Z']]

a list comprehension

[c | c<-x, c `elem` ['A'..'Z']]

c <- x is a **generator**
c is a **pattern**
  to be matched from the elements of the list x
  to be successively bound to the elements of the input list x

c `elem` ['A'..'Z']

is a **predicate** which is applied to each successive binding of c inside the comprehension
an element of the input only appears in the output list if it passes this predicate.

https://stackoverflow.com/questions/35198897/does-mean-assigning-a-variable-in-haskell

Assignment in Haskell

Assignment in Haskell : declaration with initialization:

- You declare a variable;
- Haskell doesn't allow uninitialized variables,
  - so an initial value must be supplied in the declaration
- There's no mutation, so the value given in the declaration
  - will be the only value for that variable throughout its scope.

https://stackoverflow.com/questions/35198897/does-mean-assigning-a-variable-in-haskell

Assignment in Haskell

```
filter (`elem` ['A' .. 'Z']) x
```

```
[c| c <- x]
```

```
do c <- x
   return c
```

```
x >>= \c -> return c
```

```
x >>= return
```

https://stackoverflow.com/questions/35198897/does-mean-assigning-a-variable-in-haskell

Anonymous Functions

```
(\x -> x + 1) 4
5 :: Integer


(\x y -> x + y) 3 5
8 :: Integer

addOne = \x -> x + 1


addOneList lst = map addOne' lst
  where addOne' x = x + 1

addOneList' lst = map (\x -> x + 1) lst

addOneList'' = map (+1)
```

https://wiki.haskell.org/Anonymous_function

Monad Class Function >>= & >>

both >>= and >> are functions from the Monad class.

>>= **passes** the result of the expression on the left
as an argument to the expression on the right,
in a way that respects the context the argument and function use

>> is used to **order** the evaluation of expressions within some context;
it makes evaluation of the right depend on the evaluation of the left

https://www.quora.com/What-do-the-symbols-and-mean-in-haskell

Monad – List Comprehension Examples

```
[x*2 | x<-[1..10], odd x]
```

```
do
   x <- [1..10]
   if odd x
       then [x*2]
       else []
```

```
[1..10] >>= (\x -> if odd x then [x*2] else [])
```

https://stackoverflow.com/questions/44965/what-is-a-monad

Monad – I/O Examples

```
do
   putStrLn "What is your name?"
   name <- getLine
   putStrLn ("Welcome, " ++ name ++ "!")
```

https://stackoverflow.com/questions/44965/what-is-a-monad

Monad – A Parser Example

```
parseExpr = parseString <|> parseNumber

parseString = do
        char '"'
        x <- many (noneOf "\"")
        char '"'
        return (StringValue x)

parseNumber = do
    num <- many1 digit
    return (NumberValue (read num))
```

https://stackoverflow.com/questions/44965/what-is-a-monad

Monad – Asynchronous Examples

```
let AsyncHttp(url:string) =
    async {  let req = WebRequest.Create(url)
             let! rsp = req.GetResponseAsync()
             use stream = rsp.GetResponseStream()
             use reader = new System.IO.StreamReader(stream)
             return reader.ReadToEnd() }
```

https://stackoverflow.com/questions/44965/what-is-a-monad

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf