Монады

Стрелка Клейсли: a -> m b

Функции с эффектами могут:

a -> Maybe b -- завершиться неудачей

a -> [] b -- возвращать много результатов

a -> (Either s) b -- завершиться типизированным исключением

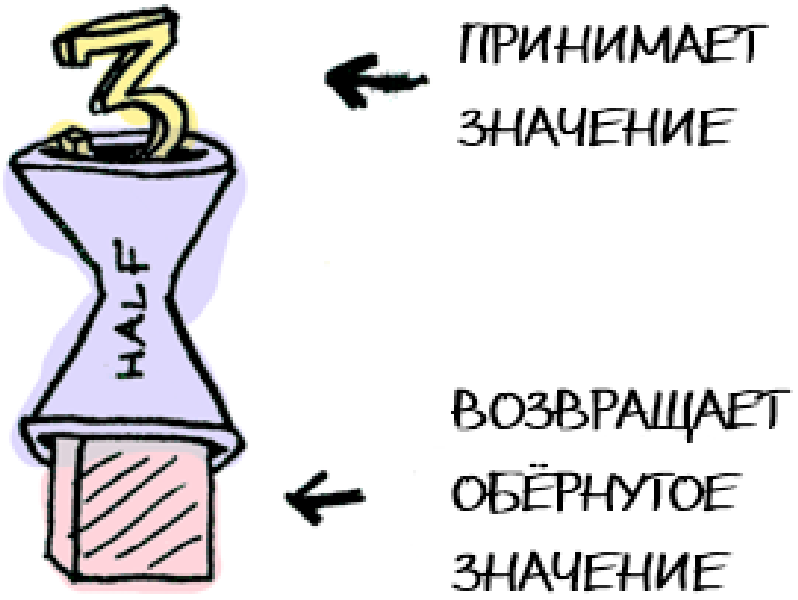
a -> ((,) w) b -- делать записи в лог

a -> ((->) e) b -- читать из внешнего окружения

a -> (State s) b -- работать с изменяемым состоянием

a -> IO b -- осуществлять ввод/вывод

Функции с эффектами



https://habr.com/ru/post/183150/

Прогулка по канату

```
type Birds = Int
type Pole = (Birds, Birds)
landLeft' :: Birds -> Pole -> Pole
landLeft' n (left, right) = (left + n, right)
landRight' :: Birds -> Pole -> Pole
landRight' n (left, right) = (left, right + n)
(|>) :: a -> (a -> b) -> b
x |> f = f x
infixl 0 |>
```

Канатоходец удержит равновесие, если количество птиц на левой стороне шеста и на правой стороне шеста разнится в пределах трёх.

Миран Липовача, Изучай Haskell во имя добра!

Прогулка по канату

```
example1 = (0, 0) $> landLeft' 1
     $> landRight' 4 $> landLeft' (-1)
     $> landRight' (-2)

ghci>example1
(0,2)
```

Что-то не так…

Миран Липовача, Изучай Haskell во имя добра!

Прогулка по канату

```
landLeft :: Birds -> Pole -> Maybe Pole
landLeft n (left,right)
  | abs ((left + n) - right) < 4 = Just (left + n, right)
  | otherwise = Nothing
landRight :: Birds -> Pole -> Maybe Pole
landRight n (left,right)
  | abs (left - (right + n)) < 4 = Just (left, right + n)
  | otherwise = Nothing
```

Миран Липовача, Изучай Haskell во имя добра!

Прогулка по канату

```
example2 = case landLeft 1 (0, 0) of
    Nothing -> Nothing
    Just pole1 -> case landRight 4 pole1 of
        Nothing -> Nothing
        Just pole2 -> case landLeft (-1) pole2 of
            Nothing -> Nothing
            Just pole3 -> landRight (-2) pole3
ghci> example2
Nothing
```

Какие минусы такого решения?

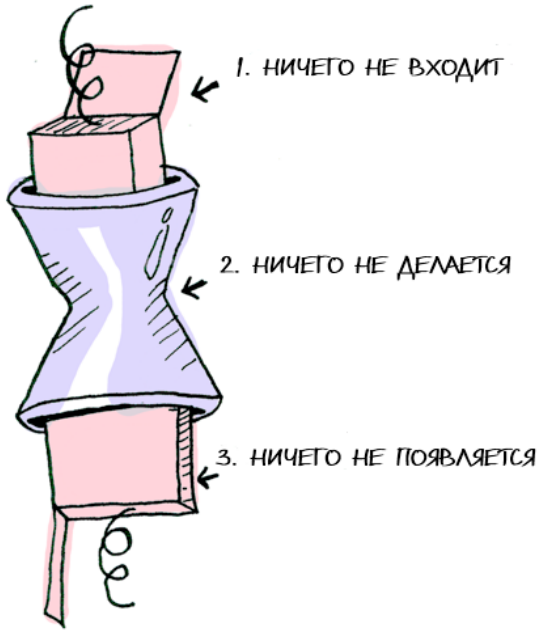
Миран Липовача, Изучай Haskell во имя добра!

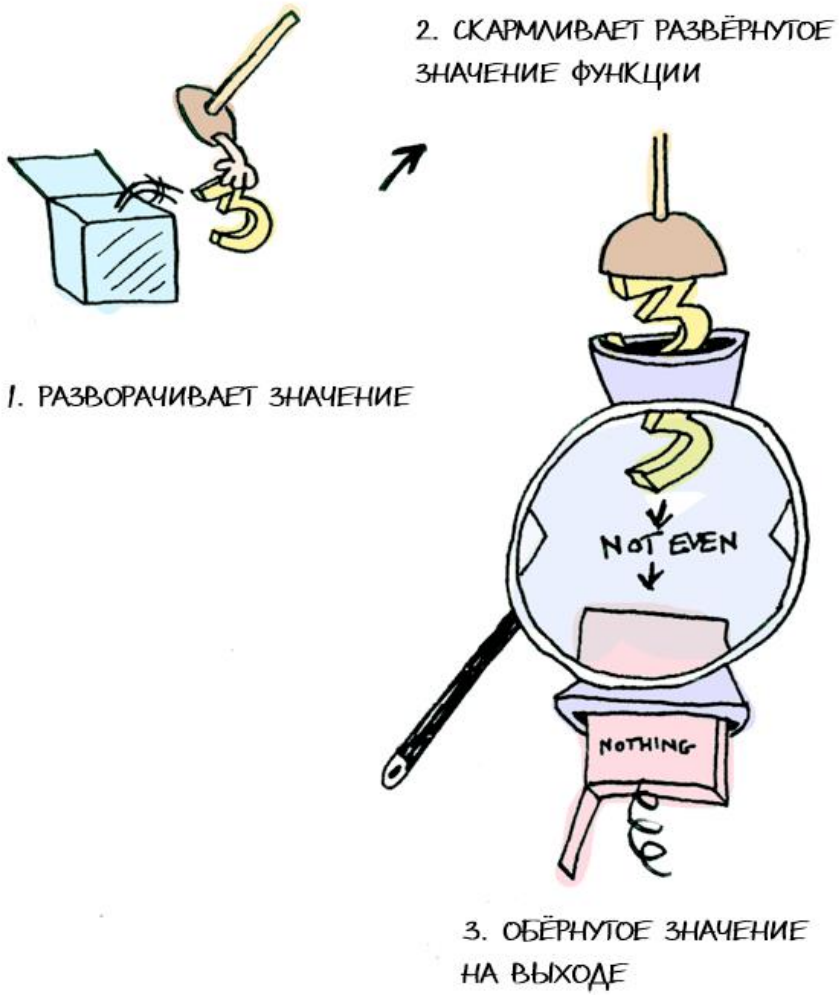
Монады

```
class Monad m where
  return :: a -> m a
  (>>=) :: m a -> (a -> m b) -> m b
  (>>) :: m a -> m b -> m b
  x >> y = x >>= \_ -> y
```

Maybe

```
instance Monad Maybe where
  return x = Just x
  Nothing >>= f = Nothing
  Just x >>= f = f x
```





https://habr.com/ru/post/183150/

Прогулка по канату

```
example3 = landLeft 1 (0, 0)
      >>= landRight 4
      >>= landLeft (-1)
      >>= landRight (-2)
```

```
ghci> example3
Nothing
```

Миран Липовача, Изучай Haskell во имя добра!

do нотация

```
do {e}                          ≡ e
do {e; stmts}                   ≡ e >> do {stmts}
do {p <- e; stmts}              ≡ e >>= \p -> do {stmts}
do {let v = exp; stmts}         ≡ let v = exp in do {stmts}
```

do нотация

```
foo :: Maybe String
foo = Just 3 >>= (\x ->
     Just "!">>= (\y ->
     Just (show x ++ y)))

foo :: Maybe String
foo = do
     x <- Just 3
     y <- Just "!"
     Just (show x ++ y)
```

Упражнение

Реализуйте **eval**, используя:

1. do-нотацию
2. оператор (>>=)

```
data Expr = Var |
            Num Integer |
            Add Expr Expr |
            Sub Expr Expr

eval :: Expr -> Maybe Integer
eval Var = Nothing
eval (Num n) = Just n
eval (Add e1 e2) = undefined
eval (Sub e1 e2) = undefined
```

Упражнение

```
data Person = Person { name :: String, surname :: String }
    deriving (Show, Eq)
grades :: Map.Map Integer Integer
grades = Map.fromList [(124001, 2), (124002, 4)]
studentIds :: Map.Map String Integer
studentIds = Map.fromList [("Ivanov I.", 124001), ("Petrov P.", 124002),
                                                  ("Sidorova S.", 124003)]
parse :: String -> Maybe Person
parse s = case words s of
    [name, surname] -> Just (Person name surname)
    _ -> Nothing
checkGrade :: String -> Maybe Integer
checkGrade = undefined

ghci> checkGrade "sjflasjf;als"
Nothing
ghci> checkGrade "Sveta Sidorova"
Nothing
ghci> checkGrade "Petya Petrov"
Just 4
```

Упражнение

```
safeDiv :: Int -> Int -> Maybe Int
safeDiv _ 0 = Nothing
safeDiv x y = Just (x `div` y)
```

Реализуйте функцию myForM, которая применяет функцию к каждому элементу списка:

```
ghci> myForM [1, 2, 3] (safeDiv 6)
Just [6,3,2]
ghci> myForM [1, 2, 0, 3] (safeDiv 6)
Nothing
```

Упражнение

С помощью myForM реализуйте функцию mySequence:

```
ghci> mySequence [Just 1, Just 2, Just 3]
Just [1,2,3]
ghci> mySequence [Just 1, Nothing, Just 3]
Nothing
```

Упражнение*

Реализуйте функцию myJoin, которая удаляет один уровень монадической структуры:

```
ghci> myJoin [[1, 2], [], [3]]
[1,2,3]
ghci> myJoin (Just (Just 1))
Just 1
ghci> myJoin (Just Nothing)
Nothing
ghci> myJoin Nothing
Nothing
```