Lesssons

IN SWIFT

THROUGH HASKELL

ME

» Joe Burgess

» iOS course at The Flatiron School

» 6 semesters!

WHY?!

I DON'T WANT TO BE A NOOB

NEW FEATURES!

New Questions!

New "Best Answer"

OBJECTS

You Are Biased!

THE DEEP END

Haskell for Great Good!

A Beginner's Guide

Miran Lipovača

no starch press

HOW DO I SOLVE PROBLEMS WITH FP

WITH FUNCTIONS

OBVI

More Specifically

» Splices in Heist

» Enums/Presenter in Yesod Apps

» Types as Documentation in Blaze and Yesod

» Continuations in Yesod Apps

» Dependency Injection using partial application

HEIST

- ERB for Haskell
- Pulls from the Lift web framework in Scala
- Generally used with the Snap Web Framework
- Simpler than Yesod
- More Info

General purpose templating with...

FUNCTIONS!

SPLICES

Factorial Function

```
factSplice :: Splice Snap
factSplice = do
    input <- getParamNode
    let text = T.unpack $ X.nodeText input
        n = read text :: Int
    return [X.TextNode $ T.pack $ show $ product [1..n]]
```

Using the Splice

```
bindSplice "fact" factSplice templateState
```

In your template:

```
<fact>5</fact>
```

Spits out

```
120
```

NEVER SUBCLASS

BECAUSE YOU CAN'T REALLY IN HASKELL

In Swift

```
class myView: UIView {
    let splice: () -> (String)
}
```

In Swift

```swift
let theView = myView(frame: frame) { () -> (String) in
  return "This is a test"
}
```

So What?

- No logic in our views
- Pass static values wrapped in anonymous functions { customer.name }
- Works well with MVVM
- Presenter Pattern

SPEAKING OF THE PRESENTER PATTERN

ENUMS AND FUNCTIONS

```
data SortBy
    = SortByAZ
    | SortByCountUp   -- lowest count at top
    | SortByCountDown -- highest count at top
    | SortByYearUp    -- earliest year at top
    | SortByYearDown  -- latest year at top
    deriving Eq

instance Show SortBy where
    show SortByAZ        = "a-z"
    show SortByCountUp   = "count-up"
    show SortByCountDown = "count-down"
    show SortByYearUp    = "year-up"
    show SortByYearDown  = "year-down"
```

source

Enums and Functions

```haskell
data ResourceType
    = BlogPost
    | CommunitySite
    | Dissertation
    | Documentation
    deriving (Bounded, Enum, Eq, Ord, Read, Show)

-- Describe a resource type in a short sentence.
descResourceType :: ResourceType -> Text
descResourceType BlogPost           = "Blog post"
descResourceType CommunitySite      = "Community website"
descResourceType Dissertation       = "Dissertation"
```

source

LAST EXAMPLE OF ENUMS POWER

```
data Shape = Circle Float Float Float | Rectangle Float Float Float Float
```

```
enum Shape {
  case Circle(Float,Float,Float)
  case Rectangle(Float,Float,Float,Float)
}

var myShape = Shape.Circle(5.0,5.0,5.0)
```

source

SO WHAT?

- Define Business Logic easily!
- Compile time checks for all options
- State Machines are Simple!

Business Logic in Enums

```
enum Barcode {
    case UPCA(Int, Int, Int, Int)
    case QRCode(String)
}
```

STATE MACHINES FEEL

HARD

STATE MACHINES ARE SIMPLE

State Machines

```
enum Light {
    case Off, On
    func flipedSwitch() -> Light {
        switch self {
        case On: return Light.Off
        case Off: return Light.On
        }
    }
}

var light = Light.On
light = light.flippedSwitch()
if light == .Off { println("Hello?") }
```

So many types!

Types as Documentation

```
paperInfoModalWithData :: PaperId -> Paper -> Widget
```

```
type PaperId = Int
```

source

Types as documentation

```
type Attributes = [(String, String)]
```

source

Type Lessons for Swift

- Be a bit silly
- Easier then documentation
- COMPILE TIME CHECKING

BACK TO FUNCTIONS

COMPLETION BLOCKS TO THE EXTREME

CONTINUATION

```haskell
runValidation :: Validation a -> a -> (a -> Handler b) -> Handler b
runValidation validate thing onSuccess =
    case validate thing of
        Right v -> onSuccess v
        Left es -> sendResponseStatus status400 $ object
            ["errors" .= map toJSON es]
```

source

Continuation

» Allow us to punt on what to do

```haskell
unitAttack :: Target -> IO ()
unitAttack target = do
  swingAxeBack 60
  valid <- isTargetValid target
  if valid
  then ??? target
  else sayUhOh
```

» I don't know what I'm going to want to do in ???

PUNT!

```haskell
unitAttack :: Target -> (Target -> IO ()) -> IO ()
unitAttack target todo = do
    swingAxeBack 60
    valid <- isTargetValid target
    if valid
    then todo target
    else sayUhOh
```

Dependency Injection

Partial Application!

```
add     :: Int -> Int -> Int
add x y = x + y
```

Which means

```
add :: Int -> (Int -> Int)
```

```
addOne = add 1 // addOne :: Int -> Int
```

IN SWIFT!

```
func getCustomer (customerID: CustomerID) -> Customer?

func getCustomerFromMem (source: [Cust])(custID: CustID) -> Cust?

func getCustomerFromDB (source: DBHandler)(custID: CustID) -> Cust?

let getMemCustomer = getCustomerFromMem(source: customersArray)
let foundCustomer = getMemCustomer(customerID: 2)
```

source

TID-BITS

Cool tidbit

```
-- Reuse instance ToMarkup Text
instance ToMarkup ResourceType where
    toMarkup = toMarkup . descResourceType
    preEscapedToMarkup = toMarkup . descResourceType
```

source

Inferred Types

- Used in type signatures of functions always
    - Because it's documentation!
- Pretty much never used anywhere else

Infix Notation

```haskell
elem 'a' "camogie"

'a' `elem` "camogie"
```

No more crazy operator symbols!

DERIVING

```
data LocalCopyStatus = LocalAvailable
                       | NotYet
                       | Failed
                       | Unknown
                       deriving (Show,Read,Eq,Typeable)
```

PROTOCOLS WITH BASIC IMPLEMENTATIONS

THANKS!

Joe Burgess

@jmburges