# LESSINS

# IN SWIFT

# THROUGH HASKELL

# ME

- » Joe Burgess
- » iOS course at The Flatiron School
- » 6 semesters!



# IDON'T WANT TO BE A NOOB

# REW FEATURES!

# NEW QUESTIONS!

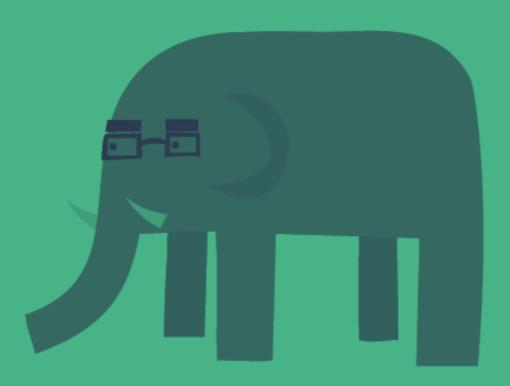
# RESUBEST ANSWER!



# You Are Biased!

# Haskell for Great Good!

A Beginner's Guide





# 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...

# FUNGIONS:

# 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]]</pre>
```

## USING THE SPLICE

bindSplice "fact" factSplice templateState

#### **IN YOUR TEMPLATE:**

<fact>5</fact>

#### **SPITS OUT**

120

# REVER SUBCLASS BECAUSE YOU CAN'T REALLY IN HASKELL

# IN SWIFT

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

# IN 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



# ENUMS AND FUNCTIONS

#### source

## ENUMS AND FUNCTIONS

data ResourceType

source

= 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"
```

# 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

# STATE MACHINES ARE SILVER STATE MACHINES ARE

# 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

# 

## CONTINUATION

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

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

### PUNT!

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

## 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

# 

## **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

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

### NO MORE CRAZY OPERATOR SYMBOLS!

## DERIVING

# PROTOCOLS WITH BASIC IMPLEMENTATIONS

Joe Burgess

@jmburges

