

Haskell

Why

Who even uses it?

Finance

Standard Chartered

>> 2.5m lines of Haskell

>> Used by engineers and traders alike to write code

Facebook

Sigma: Facebook's spam/malware/abuse detection system

- » Fighting spam with Haskell¹
- » The Road to Running Haskell at Facebook Scale – Jon Coens²

¹ <https://code.facebook.com/posts/745068642270222/fighting-spam-with-haskell/>

² <https://www.youtube.com/watch?v=sl2zo7tzt08>

Grasswire

**Austen Allred**
@AustenAllred

⚙️ [+ Follow](#)

Building in Haskell and being remote lets Grasswire play some serious moneyball. Work with really sharp people, save \$, build faster.

RETWEETS
16

LIKES
33



**Austen Allred**
@AustenAllred

⚙️ [+ Follow](#)

Seriously, making the right decisions on tech has made Grasswire's costs about 1/3 of what they would be

RETWEETS
8

LIKES
28



12:53 PM - 24 Nov 2015

What is Haskell?

Haskell is a standardized, general-purpose purely functional programming language, with non-strict semantics and strong static typing.

-- Wikipedia

What does it all
mean??

Swift

```
func sayHello(personName: String) -> String
```


Swift

```
func sayHello(personName: String) -> String {  
    let greeting = "Hello, " + personName  
    return greeting  
}
```

Swift

```
func sayHello(personName: String) -> String {  
    if arc4random() != 0 {  
        return greeting = "Hello, " + personName  
    } else {  
        return "Bye, " + personName  
    }  
    return greeting  
}
```

Swift

```
func sayHello(personName: String) -> String
```

Anything can happen 🤖

Haskell

```
sayHello :: String -> String
```

Haskell

```
sayHello :: String -> String
```

```
sayHello = ("Hello, "++)
```

Haskell

```
sayHello :: String -> IO String
```

```
sayHello name = do
```

```
    randomInt <- randomIO
```

```
    if randomInt == 0
```

```
        then return $ ("Bye, " ++ name)
```

```
        else return $ ("Hello, " ++ name)
```

Haskell

```
sayHello :: Integer -> String -> String
```

```
sayHello i name =
```

```
    if i == 0
```

```
        then return $ ("Bye, " ++ name)
```

```
        else return $ ("Hello, " ++ name)
```

```
-- Some other code/function would deal with the IO
```

TDD?

Type Driven Development

There's a saying in Haskell: "If it compiles, it works"

Type Driven Development

There's a saying in Haskell: "If it compiles, it works"

Development process ends up being:

- >> Write the types
- >> Write code until compiler stops complaining
- >> ...
- >> Profit!



Tyler Langlois

@leothrix



Follow

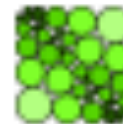
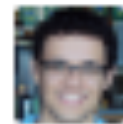
Still amazed at the “write part of the solution
-> fix compile errors -> working code”
phenomenon in Haskell

RETWEETS

7

LIKES

11



9:11 AM - 4 Dec 2015

<https://twitter.com/leothrix/status/672825545273565184>

Refactoring

"If it compiles, it works"

Testing

- >> More testable code
- >> Less things to test (especially vs. JS/Ruby/etc.)
- >> State of the art tools (QuickCheck)

Tooling

Hoogle/Hayoo/etc.

Hoogle

map

map :: (a -> b) -> [a] -> [b]
base Prelude, base Data.List
⊕ map f xs is the list obtained by applying f to each element of xs, i.e., $\text{map } f [x_1, x_2, \dots, x_n] == [f x_1, f x_2, \dots, f x_n]$

map :: (Char -> Char) -> ByteString -> ByteString
bytestring Data.ByteString.Char8, bytestring Data.ByteString.Lazy.Char8
 $O(n)$ map f xs is the ByteString obtained by applying f to each element of xs

map :: (Char -> Char) -> Text -> Text
text Data.Text, text Data.Text.Lazy
 $O(n)$ map f t is the Text obtained by applying f to each element of t. Subject to fusion. Performs replacement on invalid scalar values.

map :: (Key -> Key) -> IntSet -> IntSet
containers Data.IntSet
⊕ $O(n * \min(n, W))$. map f s is the set obtained by applying f to each element of s. It's worth noting that the size of the result may be smaller if, for some (x,y), $x \neq y$ && $f x == f y$

map :: (Word8 -> Word8) -> ByteString -> ByteString
bytestring Data.ByteString.Lazy
 $O(n)$ map f xs is the ByteString obtained by applying f to each element of xs.

Hoogle/Hayoo/etc.

Hoogle

⊖ base ⊕

⊖ parallel ⊕

(a -> b) -> [a] -> [b]

Search

(a -> b) -> [a] -> [b]

map :: (a -> b) -> [a] -> [b]

base Prelude, base Data.List

⊕ map f xs is the list obtained by applying f to each element of xs, i.e., > map f [x1, x2, ..., xn] == [f x1, f x2, ..., f xn] > map f [x1, x2, ...] == [f x1, f x2, ...]

parMap :: Strategy b -> (a -> b) -> [a] -> [b]

parallel Control.Parallel.Strategies

⊕ A combination of parList and map, encapsulating a common pattern: > parMap strat f = withStrategy (parList strat) . map f

liftA :: Applicative f => (a -> b) -> f a -> f b

base Control.Applicative

Lift a function to actions. This function may be used as a value for fmap in a Functor instance.

fmapDefault :: Traversable t => (a -> b) -> t a -> t b

base Data.Traversable

This function may be used as a value for fmap in a Functor instance.

fmap :: Functor f => (a -> b) -> f a -> f b

base Prelude, base Data.Functor, base Control.Monad, base Control.Monad.Instances

(<\$>) :: Functor f => (a -> b) -> f a -> f b

base Data.Functor, base Control.Applicative

An infix synonym for fmap.

REPL

- >> Incremental development
- >> Type inspection
- >> Type holes (`_`)

And more...

- >> Code reuse
- >> Learning curve
- >> Speed
- >> Parallelism (STM)
- >> New abstractions

How do I start?

- >> haskellbook.com
- >> CIS194
- >> #haskell/#haskell-beginners on freenode
- >> Santa Monica Haskell meetup

Thanks!