Haskell – the noCRUD way

History

- 1987 by a committee

  Lennart Augustsson, Dave Barton, Brian Boutel, Warren Burton, Joseph Fasel, Kevin Hammond, Ralf Hinze, Paul Hudak, John Hughes, Thomas Johnsson, Mark Jones, Simon Peyton Jones, John Launchbury, Erik Meijer, John Peterson, Alastair Reid, Colin Runciman, Philip Wadler

- < *Java (1995)*

Haskell agnostic

- Purely functional
- Statically typed
- Lazy

Also important and being handled today:

- Syntax in Functions
- Higher Order
- Monad

Ecosystem

- Compiler: GHC
- REPL: GHCi
- Linter: Hlint
- IDE: VIM/Emacs/ATOM/Leksah

Mainstream Comparison

- Imperative: all about state
- Functional: all about values
  - **Expression** yields **Value**
  - Every Value has an associated **type**

Code Examples

- Types
- Functions
  - Guards
  - Pattern Matching
- List Comprehensions
- Recursion

Higher Order

- A higher-order function is a function that does at least one of the following:
  - takes one or more functions as arguments
  - returns a function as its result.

- map :: (a -> b) -> [a] -> [b]
- fold(l/r) :: (b -> a -> b) -> b -> t a -> b
- filter :: (a -> Bool) -> [a] -> [a]

Algebraic Types

- data Bool = True | False
- data Student = Cruder | NoCruder

- Record Syntax

Type Classes

- ad hoc polymorphism, better known as overloading
- *Personally: look at them as interfaces*

Functor

```haskell
class Functor f where
XXXX :: (a -> b) -> f a -> f b
```

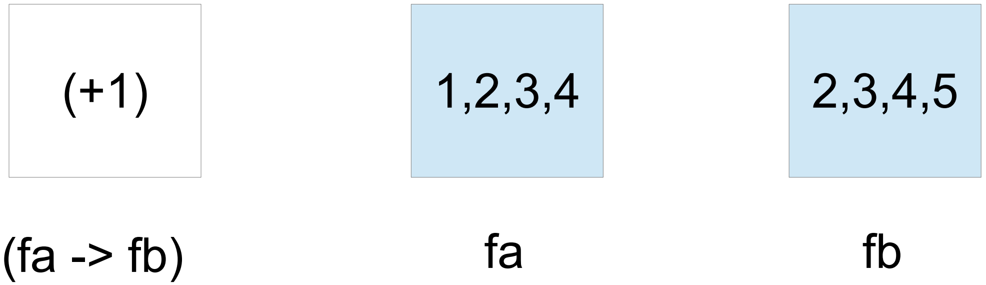
- most basic and ubiquitous type class in the Haskell libraries
- Functor represents a “container” of some sort, along with the ability to apply a function uniformly to every element in the container

Functor

```haskell
class Functor f where
XXXX :: (a -> b) -> f a -> f b
```

Example: fmap (+1)

Apply:


(+1)

1,2,3,4

2,3,4,5

(fa -> fb)

fa

fb

Example: Functor []

Applicative

```
class Functor f => Applicative f where

Pure :: a -> fa

Infixl 4 <*>

(<*>) :: f (a -> b) -> f a -> f b
```

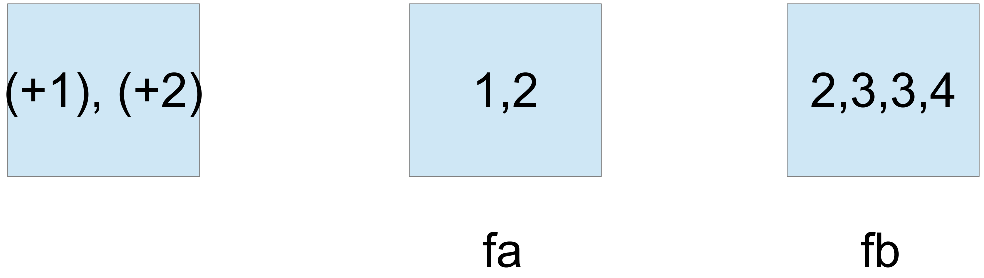
- most basic and ubiquitous type class in the Haskell libraries
- Functor represents a “container” of some sort, along with the ability to apply a function uniformly to every element in the container

Applicative

```
class Functor f => Applicative f where

Pure :: a -> fa

Infixl 4 <*>

(<*>) :: f (a -> b) -> f a -> f b
```

Example: [(+1), (+2)] <*> [1, 2]

Apply:



(+1), (+2)

1,2

2,3,3,4

fa

fb

Fact: **fmap** g x = pure g <*> x

Monoid

- Type with a rule for how two elements of that type can be combined to make another element of the same type

```
class Monoid m where
   mappend :: m -> m -> m
   mempty :: m
instance Monoid Integer where                Num a => Monoid (Sum a)
   mappend = (+)
   mempty = 0
instance Monoid Integer where                Num a => Monoid (Product a)
   mappend = (*)
   mempty = 1
```

Monad

Lazy but crazy

- Values don't need to be computed if they're not going to be used
- Infinite lists
  - used fairly often in certain areas of mathematics from what I've heard