Applicative (2A)

Young Won Lim
11/27/17

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Based on

http://learnyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Haskell in 5 steps
https://wiki.haskell.org/Haskell_in_5_steps

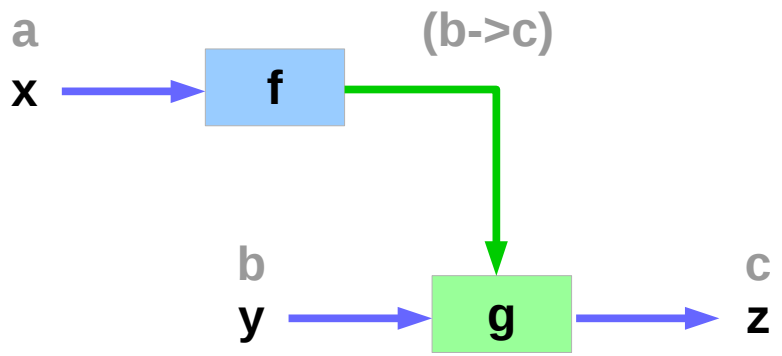
Currying

Currying recursively transforms
a function that takes multiple arguments
into a function that takes just a single argument and
returns another function if any arguments are still needed.

f :: a -> b -> c

f x y	**f :: a -> b -> c**
(f x) y	**f :: a -> (b -> c)**
g y	**g :: b -> c**

f :: a -> b -> c



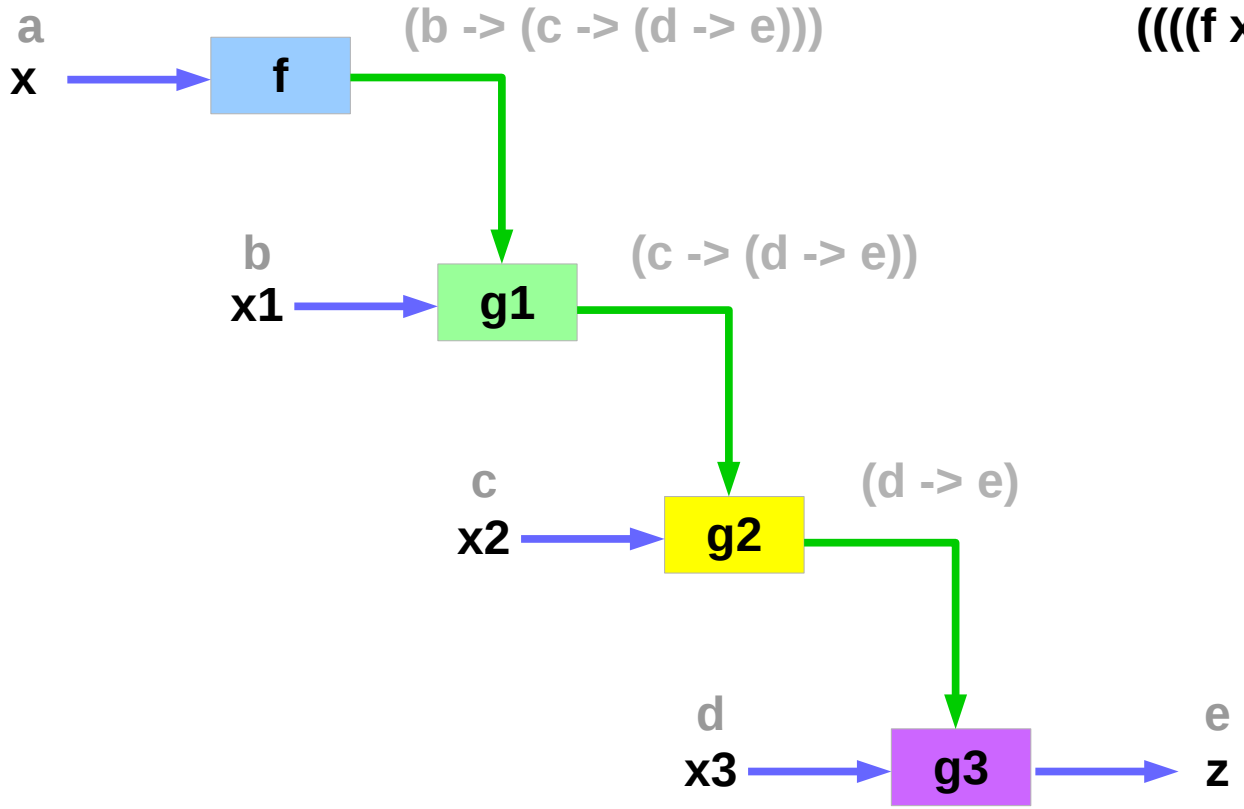
https://wiki.haskell.org/Currying
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

f :: a -> b -> c -> d -> e

f :: a -> (b -> (c -> (d -> e)))

((((f x) x1) x2) x3)



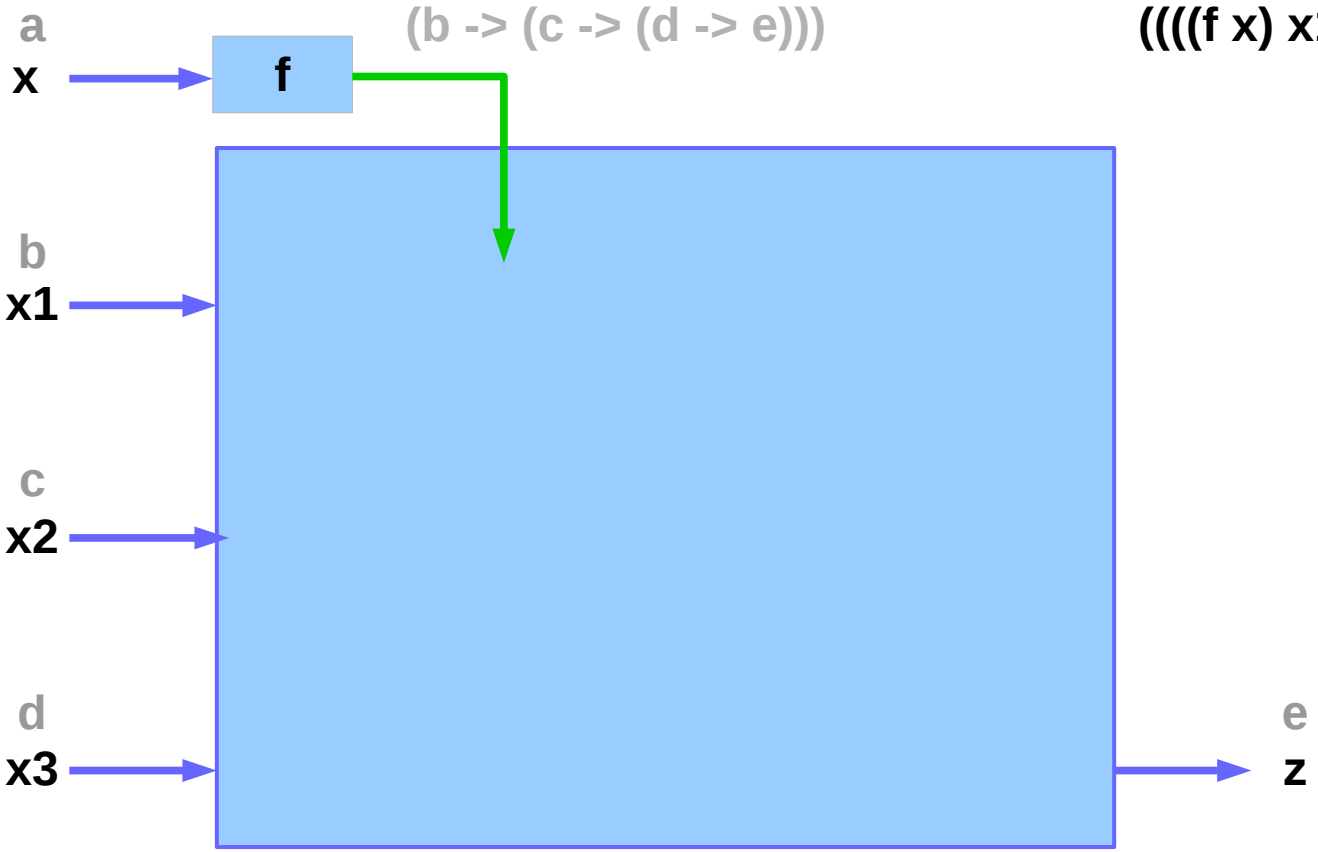
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

f :: a -> b -> c -> d -> e

f :: a -> (b -> (c -> (d -> e)))

((((f x) x1) x2) x3)



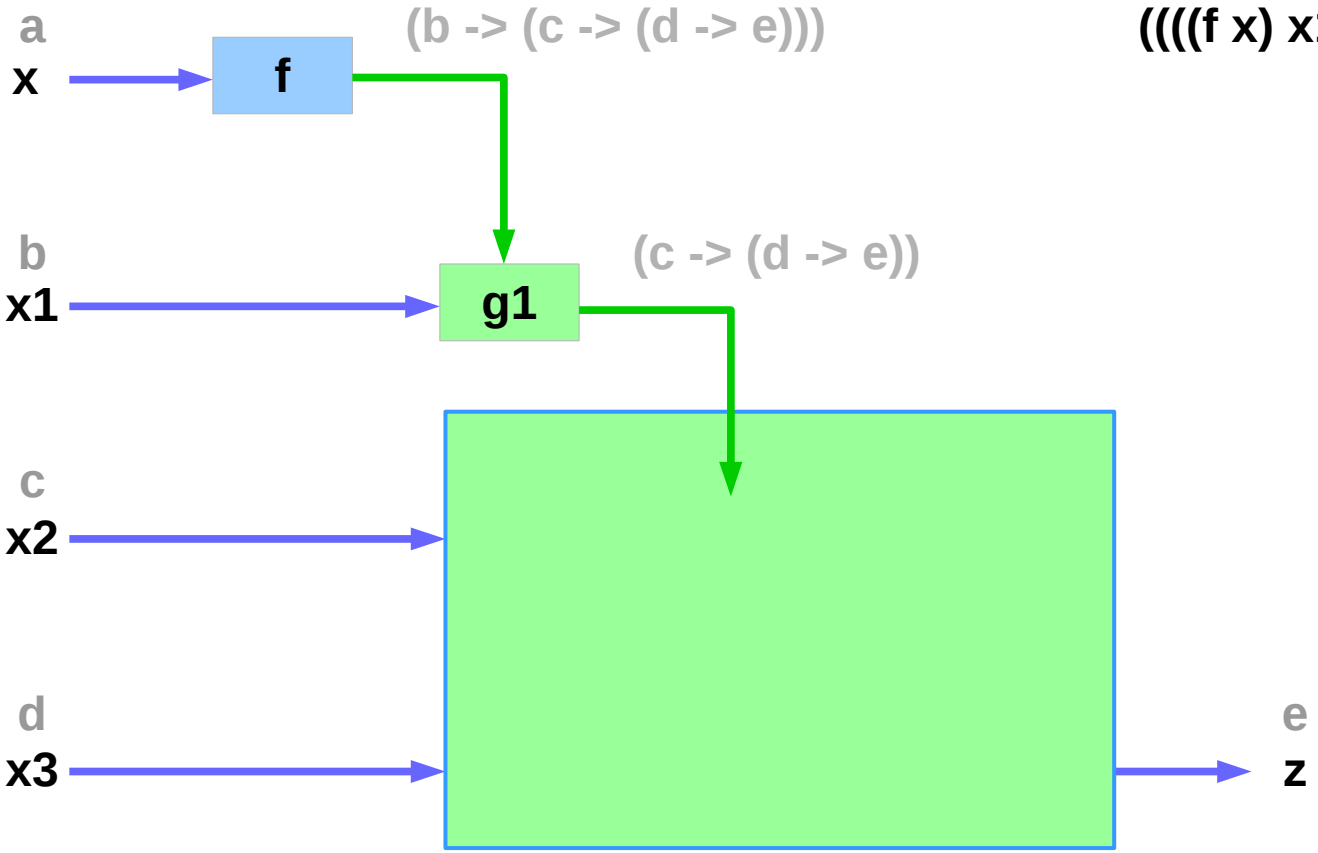
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

f :: a -> b -> c -> d -> e

f :: a -> (b -> (c -> (d -> e)))

((((f x) x1) x2) x3)



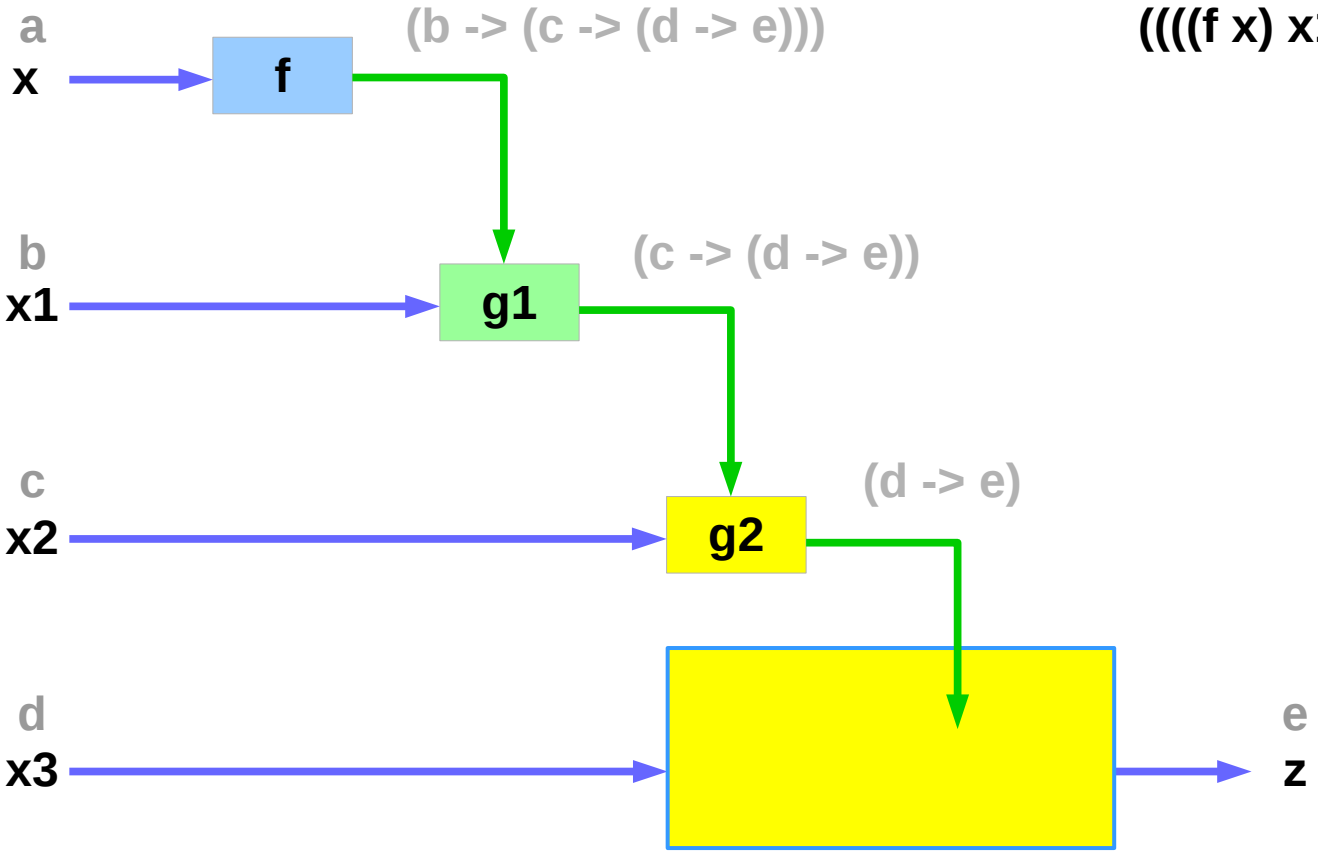
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

f :: a -> b -> c -> d -> e

f :: a -> (b -> (c -> (d -> e)))

((((f x) x1) x2) x3)



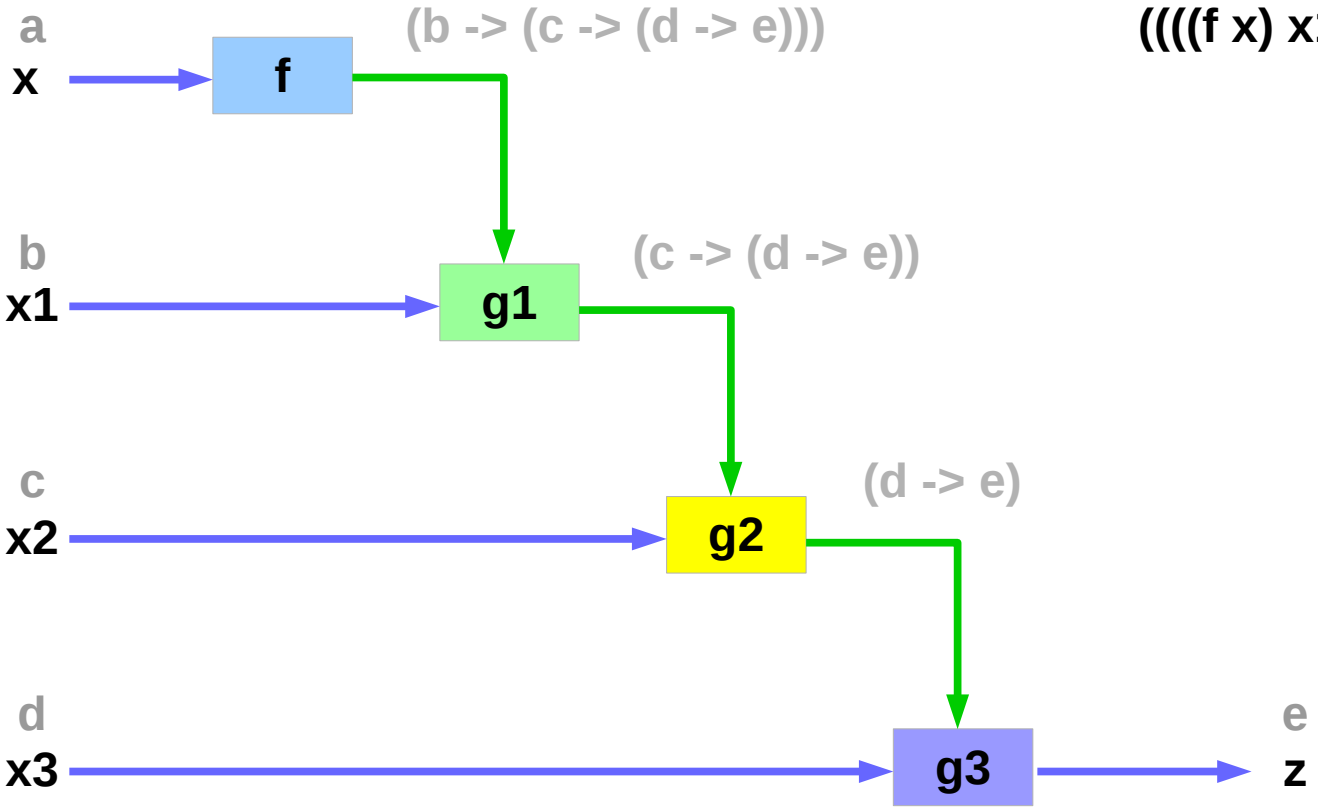
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

f :: a -> b -> c -> d -> e

f :: a -> (b -> (c -> (d -> e)))

((((f x) x1) x2) x3)



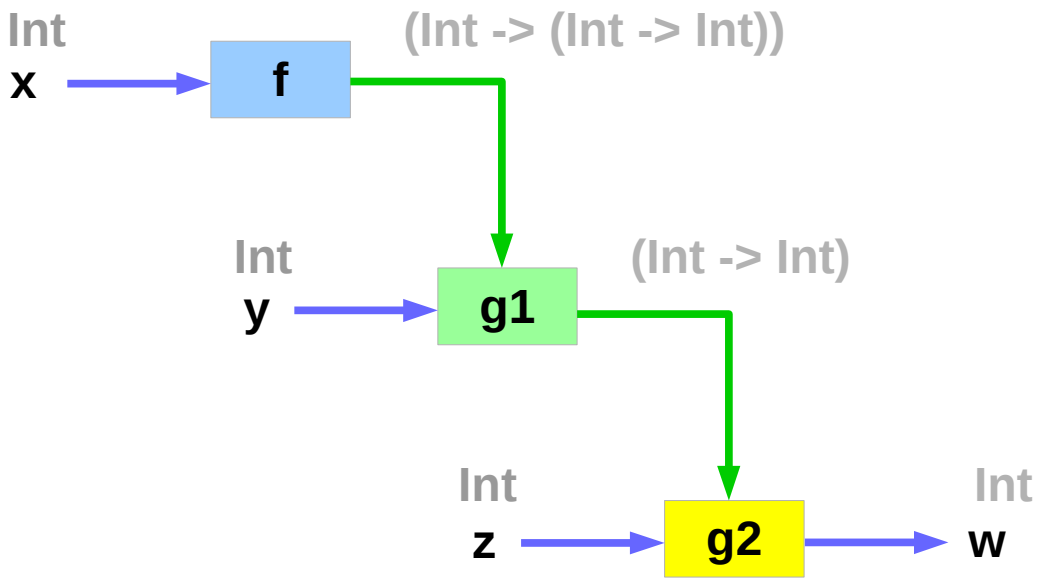
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

mult :: Int -> Int -> Int -> Int

f :: a -> (b -> (c -> (d -> e)))

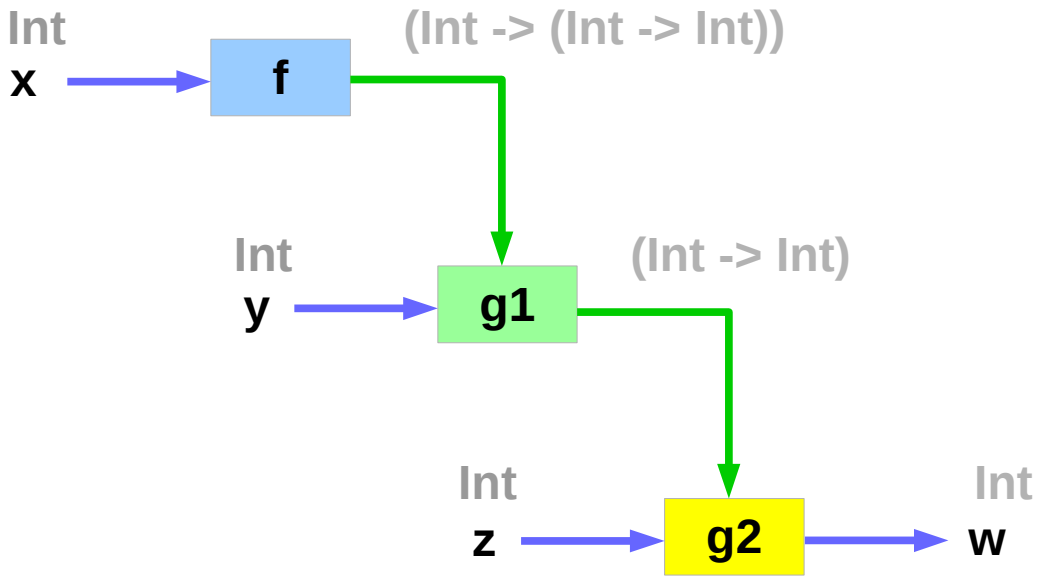
(((mult x) y) z)



http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

mult :: Int -> Int -> Int -> Int



f :: Int -> (Int -> (Int -> Int))

f x :: Int -> (Int -> Int)
g1 :: Int -> (Int -> Int)

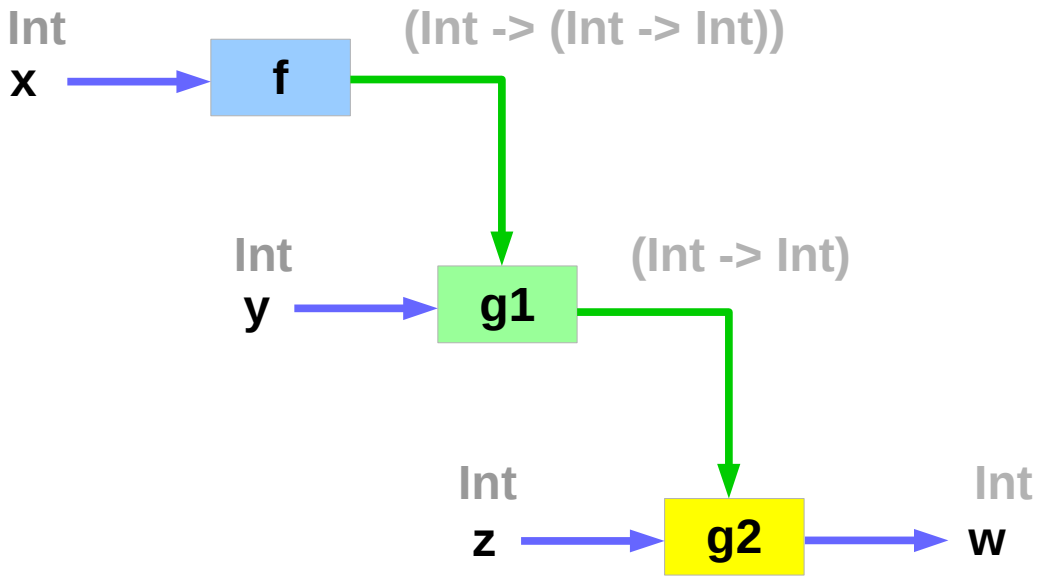
f x y :: Int -> Int
g2 :: Int -> Int

f x y z :: Int

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

mult :: Int -> Int -> Int -> Int



mult x y z

mult a_1 y z

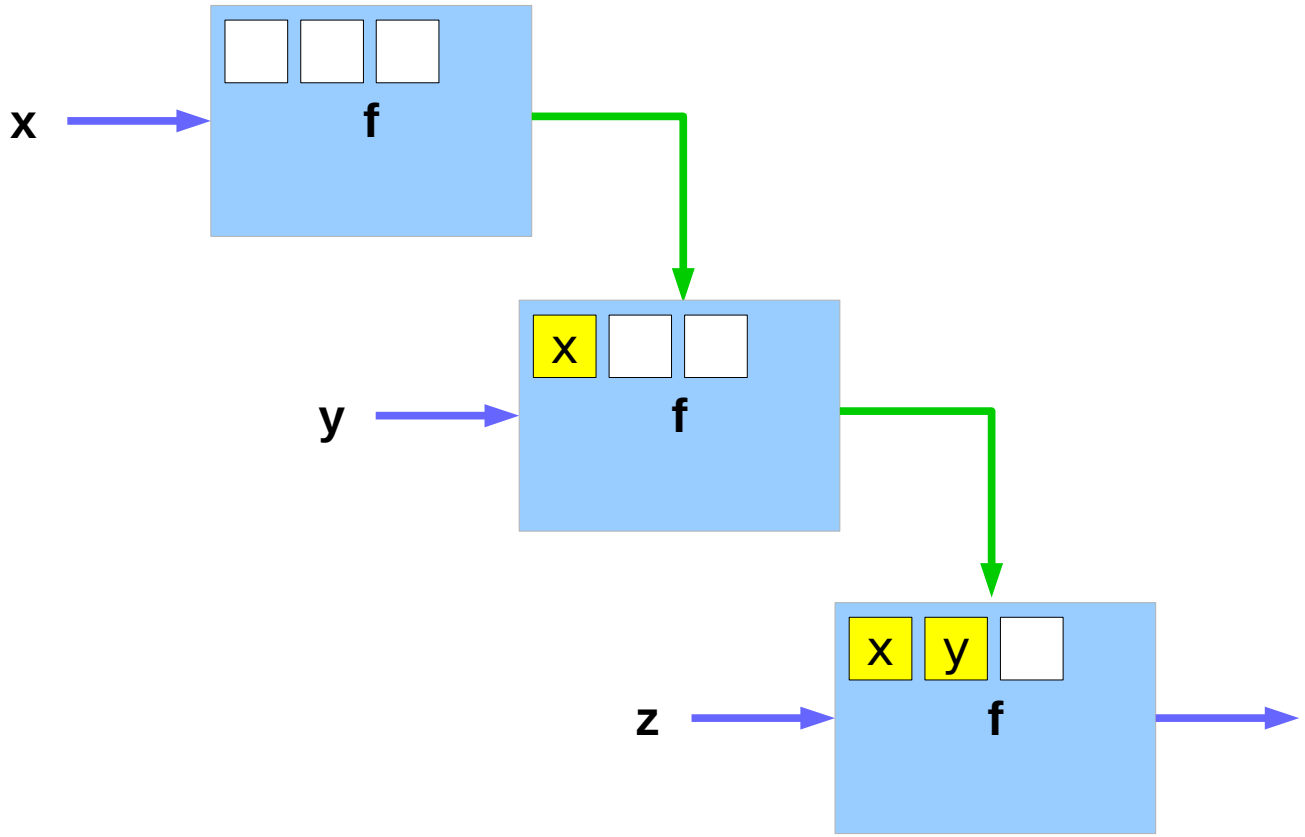
mult a_1 a_2 z

mult a_1 a_2 a_3

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying Examples

mult :: Int -> Int -> Int -> Int



mult x y z

mult a_1 y z

mult a_1 a_2 z

mult a_1 a_2 a_3

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Curry & Uncurry

f :: a -> b -> c the curried form of g :: (a, b) -> c

f = curry g
g = uncurry f

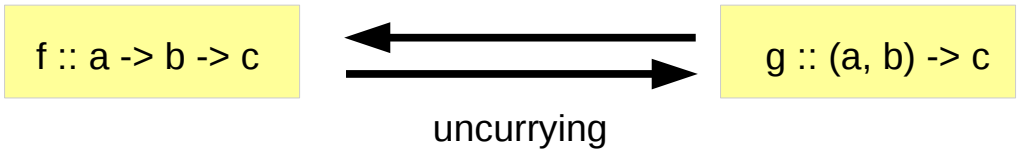
f x y = g (x,y)

the curried form is usually more convenient
because it allows partial application.

all functions are considered curried

all functions take just one argument

the curried form



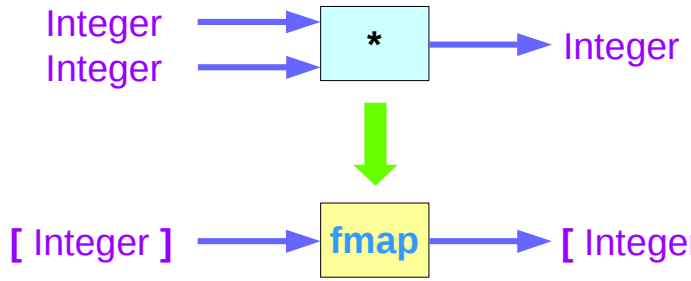
f x y

g (x,y)

https://wiki.haskell.org/Currying

Mapping functions over the Functor [ ] (1)

```
ghci> let a = fmap (*) [1,2,3,4]
```



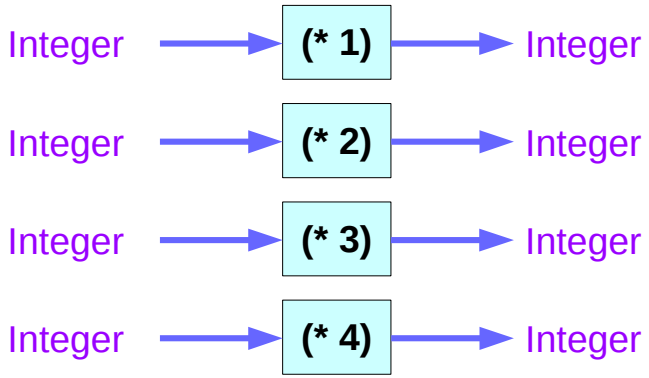
[1,2,3,4]

[(*) 1, (*) 2, (*) 3, (*) 4]

[(* 1), (* 2), (* 3), (* 4)]

[Integer -> Integer]

```
ghci> :t a
a :: [Integer -> Integer]
```



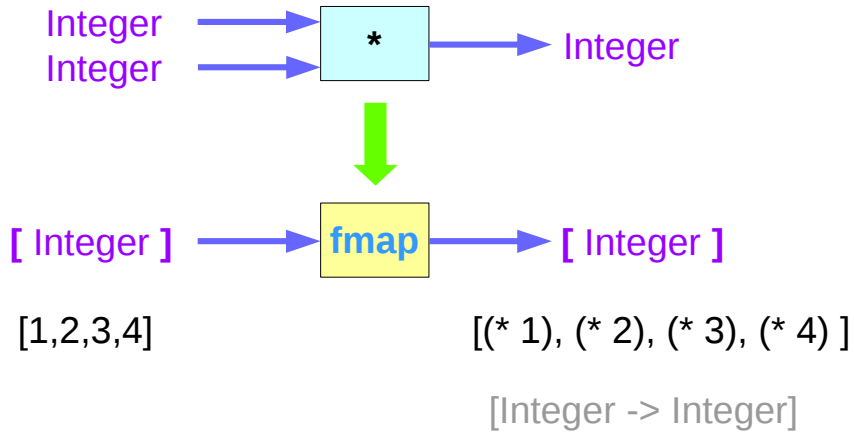

```
ghci> fmap (\f -> f 9) a
[9,18,27,36]
```

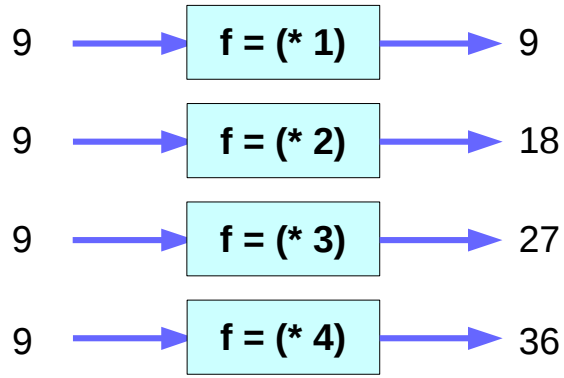
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

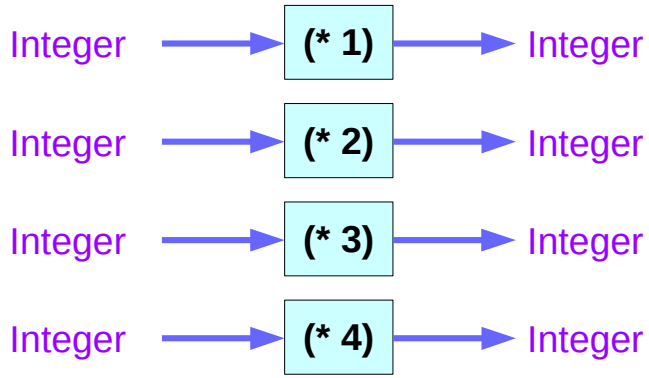
Mapping functions over the Functor [ ] (2)

```
ghci> let a = fmap (*) [1,2,3,4]
ghci> :t a
a :: [Integer -> Integer]

ghci> fmap (\f -> f 9) a
[9,18,27,36]
```



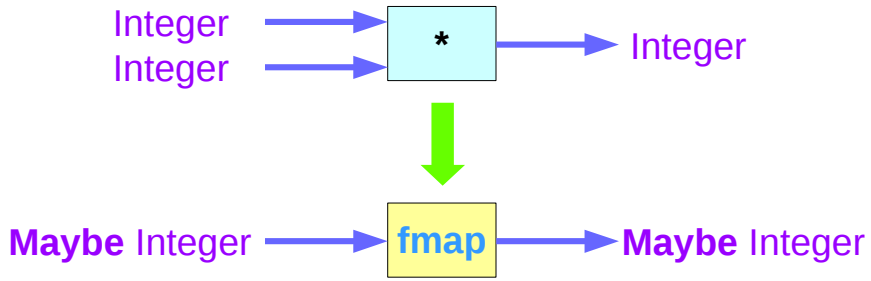




http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Mapping functions over the Functor Maybe (1)

fmap (*) (Just 3)



Just 3

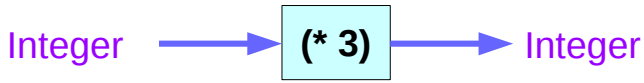
Just ((*) 3)

Just (* 3)

Maybe (Integer -> Integer)

function wrapped in a **Just**

Just (* 3)



integer wrapped in a **Just**

Just 2

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Extracting and Mapping a function

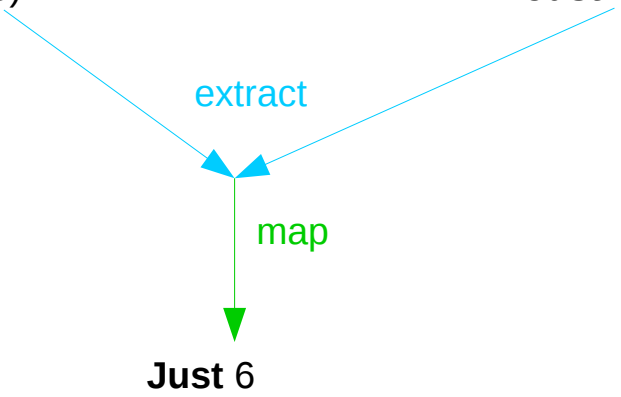
fmap (*) (Just 3)

function wrapped in a **Just**

Just (* 3)

integer wrapped in a **Just**

Just 2

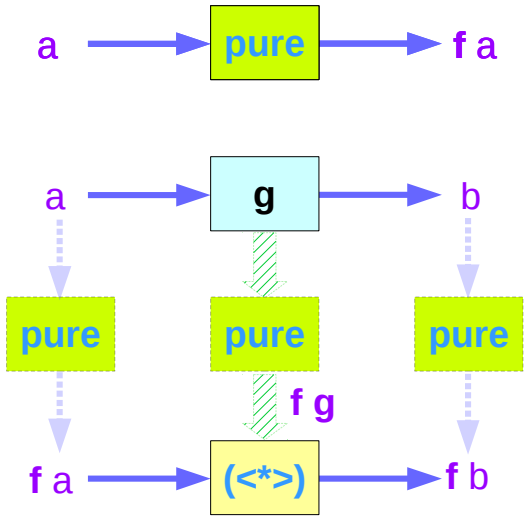


Just (* 3) <*> Just 2

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

The Applicative Typeclass

```
class (Functor f) => Applicative f where
    pure :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

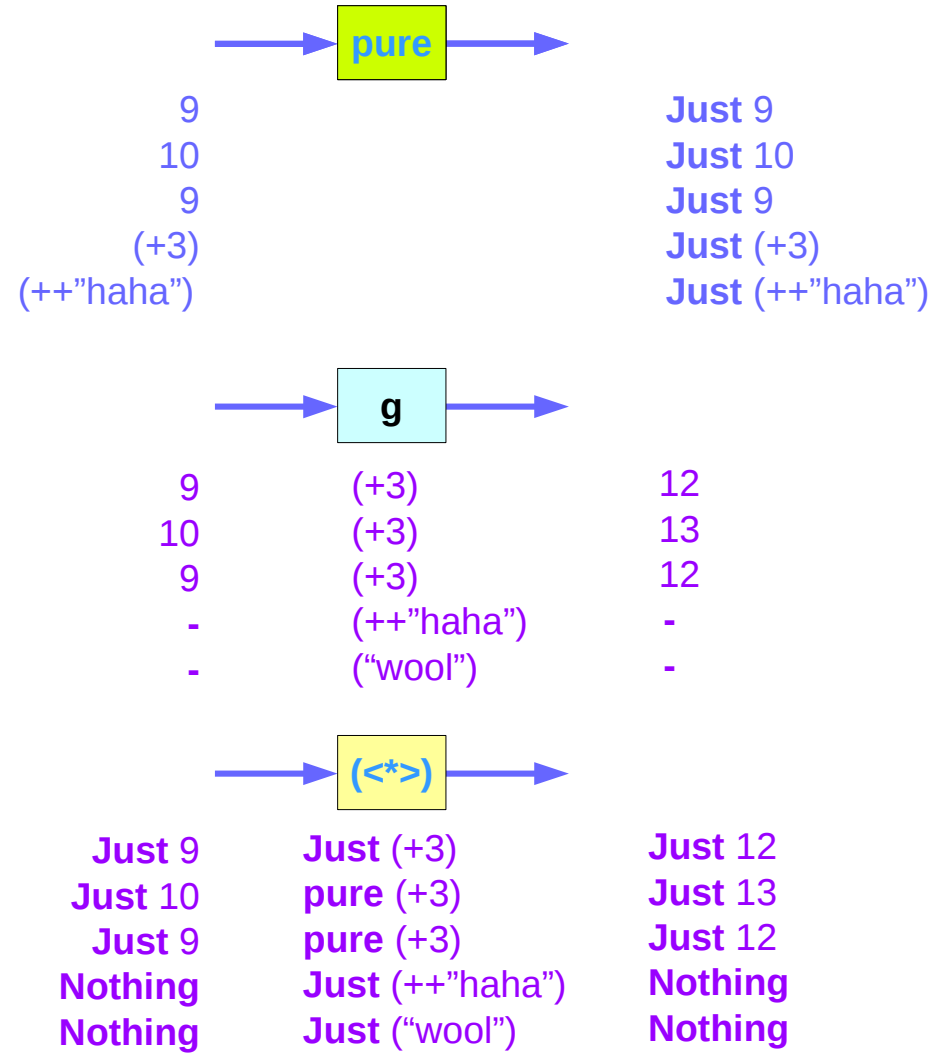



```
instance Applicative Maybe where
    pure = Just
    Nothing <*> _ = Nothing
    (Just f) <*> something = fmap f something
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

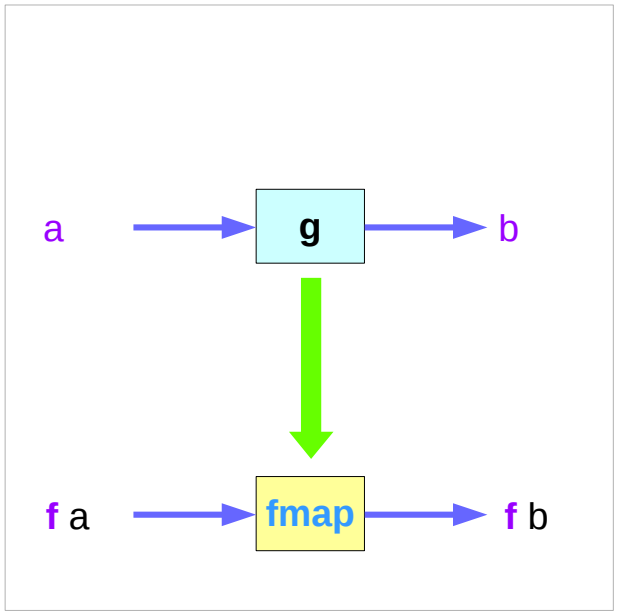
Applicative Functor Examples

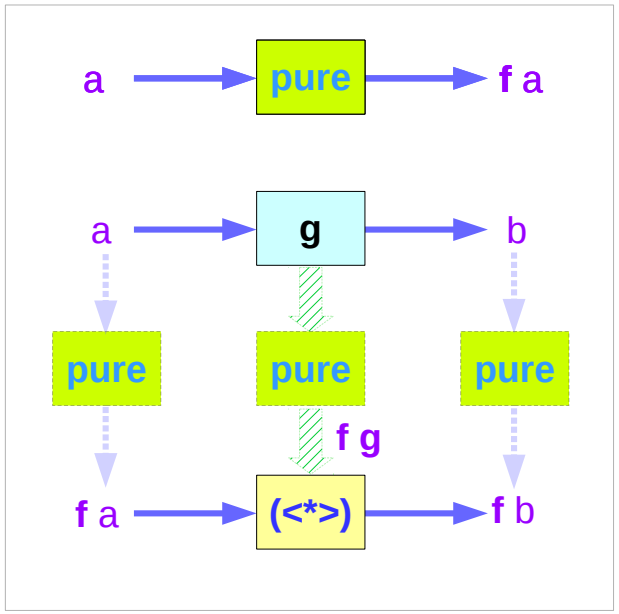
```
ghci> Just (+3) <*> Just 9
Just 12
ghci> pure (+3) <*> Just 10
Just 13
ghci> pure (+3) <*> Just 9
Just 12
ghci> Just (++"hahah") <*> Nothing
Nothing
ghci> Nothing <*> Just "woot"
Nothing
```

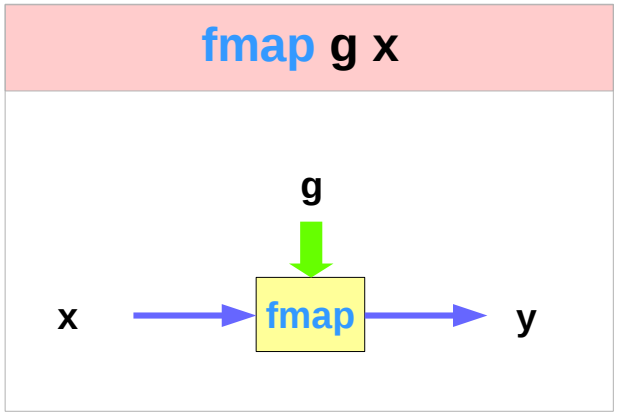


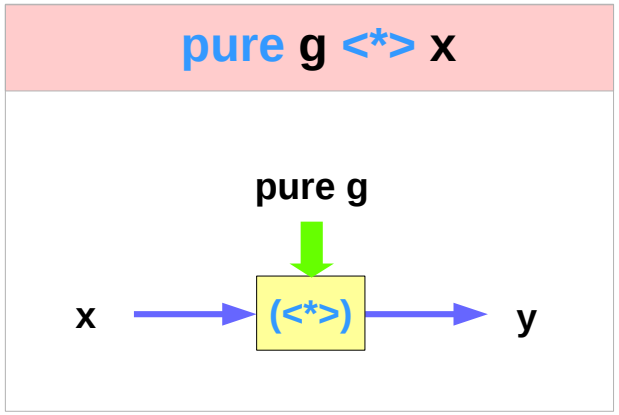
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

fmap g x = pure g <*> x





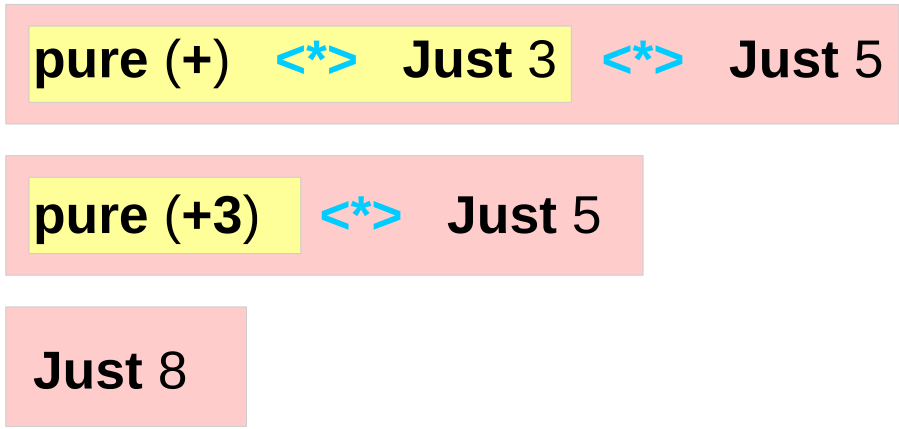




http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Left Associative <*>

```
ghci> pure (+) <*> Just 3 <*> Just 5
Just 8
ghci> pure (+) <*> Just 3 <*> Nothing
Nothing
ghci> pure (+) <*> Nothing <*> Just 5
Nothing
```



http://learnyouahaskell.com/functors-applicative-functors-and-monoids

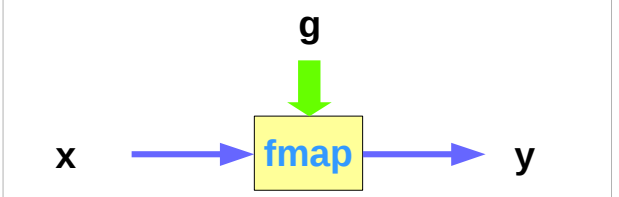
Infix Operator <$>

```
pure f <*> x <*> y <*> z
```

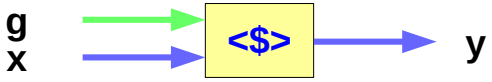
```
fmap f x <*> y <*> z
```

```
f <$> x <*> y <*> z
```

fmap g x



g <$> x



http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Infix Operator <$>

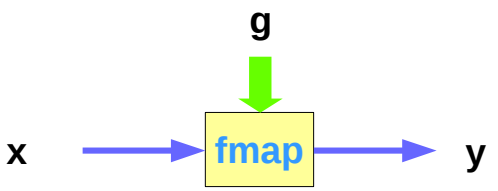
```
class (Functor f) => Applicative f where
  pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

not a class method

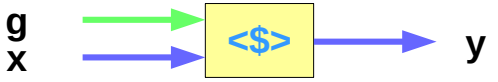
```
(<$>) :: (Functor f) => (a -> b) -> f a -> f b
f <$> x = fmap f x
```

```
instance Applicative Maybe where
  pure = Just
  Nothing <*> _ = Nothing
  (Just f) <*> something = fmap f something
```

fmap g x



g <$> x



http://learnyouahaskell.com/functors-applicative-functors-and-monoids

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf