

1 Understand Functor and Applicative Functor

1.1 Functor and Applicative

What is Applicative Functor in Haskell?

Applicative Functor is almost like Functor

Functor - it takes the value inside a box and compute it, then wrap the result into a box.

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

-- Functor needs to satisfy two laws:
-- fmap id = id
-- fmap(f . g) = (fmap f)(fmap g)

instance Functor Maybe where
    fmap :: (a -> b) -> Maybe a -> Maybe b
    fmap _ Nothing = Nothing
    fmap f (Just x) = (Just f x)

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

From the definition of Applicative Functor, If you want to be a Applicative Functor, you have to be a Functor first. In other words, Applicative is also a Functor and you need to implement two functions:

```
pure :: a -> f a
<%> :: f (a -> b) -> f a -> f b
```

Maybe is a Applicative Functor, and we know Maybe is also a Functor too, let's implement the two functions.

```
instance Applicative Maybe where
    pure a = (Just a)    -- pure = Just
```

For Applicative Maybe, If you pass anything to pure, then pure just wraps it with Just.

```
Nothing <%> _ = Nothing
_ <%> Nothing = Nothing
(Just f) <%> (Just a) = (Just f a)
```

< % > either side is Nothing then Nothing comes out < % > if f is inside Just, < % > **extracts** f out and applies fmap f (Just x) = (Just f x) How to implement < % > with fmap?

Functor contains a function ($a \rightarrow b$), fmap applies the function ($a \rightarrow b$) to a functor and wrap the result inside a functor, The difference between Functor and Applicative Functor:

Functor: < \$ > applies ($a \rightarrow b$) inside fa

Applicative Functor: < % > **extracts** ($a \rightarrow b$) out from functor and applies it inside fa

```
(+ 3) <$> (Just 2) => Just 5    -- Functor
(Just (+ 3)) <*> (Just 2) => Just 5 -- Applicative Functor
(++>) <$> getLine <*> getLine = ((++>) <$> getLine) <*> getLine => "a" ++ "b"
```