Problem 0

- The functor function is fmap. The type of fmap is:

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

- The Applicative functions are `(<*>)` and `pure`. The type of `(<*>)` is:

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

  The type of `pure` is:

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

- The Monad functions are `(>>=)` and `return`. The type of `(>>=)` is:

  ```
  (>>=) :: Monad m => m a -> (a -> m b) -> m b
  ```

  The type of `return` is:

  ```
  return :: Monad m => a -> m a
  ```

Problem 1

To explain my solution, `pure` makes an applicative functor that is then applied to the set of fs. `<*>` takes a functor that itself contains a functor, then applies that functor to the rest of the fs.

Problem 2

`myAp` works by taking in two arguments, applying left argument to the right (via the identity function), then lifting it to the applicative functor level.

The general idea of `liftA2` is to lift a functor to an applicative functor level. I implemented this through essentially creating an either structure- the `<*>` applies

Problem 3

`join'` works by applying the id function, which does nothing, and then returns the effect of the monad.

`bind'` however, works by lifting applicative functors to the monad level, then applying join to that structure.