"Functors"

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

<$> = fmap

f <$> [x] = [y]

= [fx]

Maybe

f <$> Nothing = Nothing

f <$> [x] = [fx]

(Just x) (Just $ fx)

List

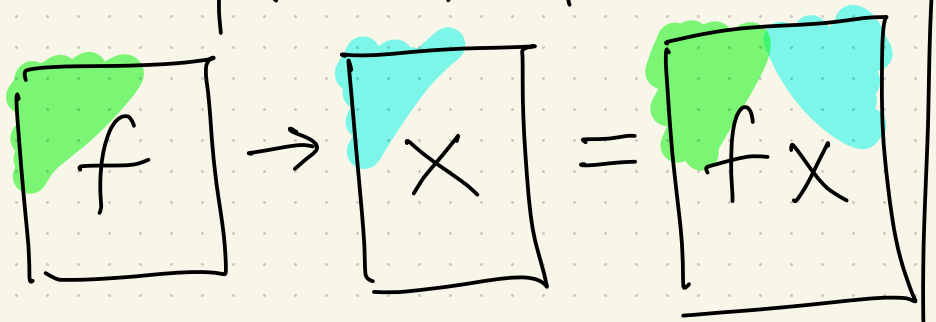
f <$> [] = []

f <$> [x, y, z] = [fx, fy, fz]

fmap = map

"Applicatives"

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



option 2:

$[f_1x_1, f_1x_2, f_1x_3, \ldots, f_2x_1, f_2x_2, f_2x_3, \ldots]$

Maybe:

$\emptyset$ <*> _ = $\emptyset$

_ <*> $\emptyset$ = $\emptyset$

Just f <*> Just x = Just $f x$

List

[] <*> _ = []

_ <*> [] = []

$[f_1, f_2, f_3 \ldots]$ <*> $[x_1, x_2 x_3, \ldots]$

= ?

option 1:

$= [f_1x_1, f_2x_2, f_3x_3, \ldots]$

"Monads"

```
class (Applicative m) => Monad m where
  return :: a -> m a
  return = pure
  (>>=) :: m a -> (a -> m b) -> m b
  (>>) :: m a -> m b -> m b
```

"bind"

"sequence"