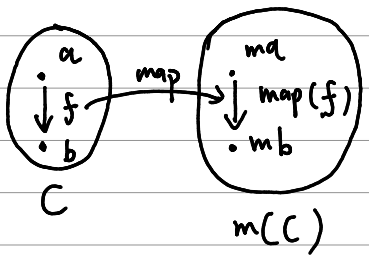
```
class Functor m where
  map :: (a -> b) -> m a -> m b
```

map lifts a function in "primitive category" to new category



C → m(C)

↑ represent the "context" of this computation with extra logic bundled.

what about higher-order functions?

eg. g :: a -> b -> c -> d

given {ma, mb, mc}, we want md.

function signature should be ma -> mb -> mc -> md.

With only 'map' in hand,

map (a->b->c->d) = m a -> m (b -> c -> d)

then populate the first argument with ma,

map (a->b->c->d) ma = m (b -> c -> d) ← "unwrap" the function inside.

Cannot proceed, need to somehow

idea: given {ma, mb, mc}, a->b->c->d —map→ ma -> m(b->c->d)

ma → m(b->c->d) —?→ (b -> c -> d) —map→ mb -> m(c->d)

mb → m(c->d) —?→ (c -> d) —map→ mc -> md

mc → md

out of "context", not composable

class Functor m ⇐ Apply m where

apply :: m a → m (a → b) → m b

partial application, apply (ma) :: m (a → b) → m b

given {ma, mb, mc}, a → b → c → d $\xrightarrow{map}$ ma → m (b → c → d)

$\xrightarrow{ma}$ m (b → c → d) $\xrightarrow{apply\ (mb)}$ m (c → d) $\xrightarrow{apply\ (mc)}$ m d

"Unify" map and apply

map :: (a → b) → ma → mb

flip (map) :: ma → (a → b) → m b

apply :: ma → m (a → b) → mb

a → b → c → d $\xrightarrow{flip(map)\ (ma)}$ m (b → c → d) $\xrightarrow{apply\ (mb)}$ m (c → d) $\xrightarrow{apply\ (mc)}$ m d

class Apply m ⇐ Applicative m where

pure :: a → m a

a → b → c → d $\xrightarrow{pure}$ m (a → b → c → d) $\xrightarrow{apply\ (ma)}$ $\xrightarrow{apply\ (mb)}$ $\xrightarrow{apply\ (mc)}$ m d

Functor (map)

↑↑

Apply (apply)

↑↑

Applicative (pure)

(a → b) → (a → m b)

λ a f. pure (f a)

another way to define 'apply'

$a \to b \to c \to d \xrightarrow{map} m\,a \to m\,(b \to c \to d) \xrightarrow{ma} m\,(b \to c \to d) \xrightarrow{?}$

Without directly "unwrap" the function out of "context",
we can "lift" a value into the "context".

value and function are duals:

apply a value to a function $\cong$ apply a function to a value

$$(a \to b) \xrightarrow{a} b \quad \xleftrightarrow{\text{isomorphism}} \quad a \xrightarrow{(a \to b)} b$$

map ::
$(a \to b) \to (m\,a \to m\,b)$

map* (dual of map) ::
$a \to (m\,(a \to b) \to m\,b)$

In previous example,

$$m\,(b \to c \to d) \xrightarrow{map^*\,(b)} m\,(c \to d) \xrightarrow{map^*\,(c)} m\,d$$

map* can be another minimum requirement
for Type Class 'Apply'.

```
class Apply m where
  map* :: a -> m (a -> b) -> m b
```