framp.me

understanding Profunctors

Functor

a Functor is a container with a `.map()`

```
const List = (a) => ({
  value: a,
  // map :: List a -> (a -> b) -> List b
  map: (fn) => List(a.map(fn))
})

assert.deepEqual(
  List([1,2]).map(a => a*2).value,
  List([2,4]).value)
```

a Functor is a container with a `.map()`

```
// Maybe :: Just a | Nothing

const Just = (a) => ({
  value: a,
  // map :: Maybe a -> (a -> b) -> Maybe b
  map: (fn) => List(a.map(fn))
})
const Nothing = {
  map: (fn) => Nothing
}

assert.deepEqual(
  Just(1).map(a => a*2).value,
  Just(2).value)

assert.deepEqual(
  Nothing.map(a => a*2).value,
  Nothing)
```

a Functor is a container with a `.map()`

can a function be a Functor?

```
const Func = (f) => ({
  value: f,
  // map :: Func a -> (a -> b) -> Func b
  map: (fn) => Func((...args) => fn(f(...args)))
})
const Nothing = {
  map: (fn) => Nothing
}

assert.deepEqual(
  Func(a => a+4)
    .map(a => a*2)
    .map(a => a*3).value(1), 30)
```

Bifunctor

a Bifunctor is a container with a `.bimap()`

```
const Pair = (a, b) => ({
  values: [a, b],
  //bimap :: Pair a b -> (a -> c) -> (b -> d) -> Pair c d
  bimap: (fnA, fnB) => Pair(fnA(a), fnB(b))
})

assert.deepEqual(
  Pair(1, 2).bimap(a => a*2, a => a*3).values, [2, 6])
```

Bifunctor

a Bifunctor is a container with a `.bimap()`

is a Bifunctor a Functor?

```
const Pair = (a, b) => ({
  values: [a, b],
  //bimap :: Pair a b -> (a -> c) -> (b -> d) -> Pair c d
  bimap: (fnA, fnB) => Pair(fnA(a), fnB(b)),
  //first :: Pair a b -> (a -> c) -> Pair c b
  first: (fn) => Pair(fn(a), b),
  //second :: Pair a b -> (b -> d) -> Pair a d
  second: (fn) => Pair(a, fn(b))
})

assert.deepEqual(
  Pair(1, 2).first(a => a*2).values, [2, 2])
```

a Bifunctor is a container with a `.bimap()`

can a function be a Bifunctor?

```
const Func = (f) => ({
  value: f,
  // bimap :: Func a b -> (a->c) -> (b->d) -> Func c d
  bimap: ???
})
```

Fake code:

```
// length :: Func String Int
const length = Func(a => a.length)
// transformOutput :: Int -> Float
const transformOutput = b => b / 3
// transformInput :: String -> Bool
const transformInput = a => a === 'banana'
// out :: Func a -> d ???
const out = length.bimap(transformInput, transformOutput)
```

Contravariant

a Contravariant is a container with a `.contramap()`

```
const Predicate = (f) => ({
  value: f,
  // contramap :: Predicate a -> (b->a) -> Predicate b
  contramap: (fn) => Predicate((...args) => f(fn(...args)
})

assert.deepEqual(
  Predicate(a => a > 5).value(6), true)

assert.deepEqual(
  Predicate(a => a > 5)
    .contramap(a => a.length).value('cat'), false)
```

opposed to Functor's map: Functor a -> (a->b) -> Functor b

```
map: (fn) => Func((...args) => fn(f(...args)))
```

in jargon:

Functors map covariantly ***or***

Functors exhibit covariance

```
//Functor a -> (a->b) -> Functor b
```

Contravariants map contravariantly ***or***

Contravariants exhibit contravariance

```
//Contravariant a -> (b->a) -> Contravariant b
```

a Contravariant is a container with a `.contramap()`

can a function be a Contravariant?

```
const Func = (f) => ({
  value: f,
  // contramap :: Func a -> (b -> a) -> Func b
  contramap: (fn) => Func((...args) => f(fn(...args)))
})

assert.deepEqual(
  Func(a => a+4)
    .contramap(a => a*2)
    .contramap(a => a+3).value(1), 12)
```

Profunctor

a Profunctor is a Bifunctor contravariant in the first argument `

a Profunctor is a container with a `.promap()`

```
const Func = (f) => ({
  value: f,
  // promap :: Func a b -> (c->a) -> (b->d) -> Func c d
  promap: (contra, co) =>
    Func((...args) => co(f(contra(...args))))
})

assert.deepEqual(Func(a => a*2)
        .promap(a => a.length, a => a*4).value('ab'), 16)
```

Fin.

slides on framp.me/profunctors