Applicatives

pure

`pure x` puts `x` into a functor (a container).

(<*>)

Called "ap".
Left associative by default.
You can also put functions inside functors: `Just succ`
How to apply such functions?

Definitions

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

```
instance Applicative Maybe where
  pure :: a -> Maybe a
  pure x = Just x
  (<*>) :: Maybe (a -> b) -> Maybe a -> Maybe b
  Just f <*> Just x = Just (f x)
  _ <*> _ = Nothing

instance Applicative [] where
  pure :: a -> [a]
  pure x = [x]
  (<*>) :: [a -> b] -> [a] -> [b]
  (f : fs) <*> xs = map f xs ++ fs <*> xs
  _ <*> _ = []
```

Rules

A valid instance of an Applicative must satisfy these laws:

1. Homomorphism: `pure f <*> pure x = pure (f x)`
2. Identity: `pure id <*> v = v`
3. Composition: `pure (.) <*> u <*> v <*> w = u <*> (v <*> w)`
4. Interchange: `u <*> pure x = pure ($ x) <*> u`