Class 7: Type Classes

March 12

review: parametricity

```
f :: a -> a -> a
f a1 a2 = a1 && a2
                 ✗
```

```
f :: a -> a -> a    ✗
f a1 a2 = case (typeOf a1) of
  Int -> a1 + a2
  Bool -> a1 && a2
  _ -> a1
```

```
f :: a -> a -> a
```

parametric polymorphism

```haskell
f :: a -> a -> a
f a1 a2 = a1

f :: a -> a -> a
f a1 a2 = a2
```

parametric polymorphism

type classes

```
(+) :: Num a => a -> a -> a
(==) :: Eq a => a -> a -> Bool
(<) :: Ord a => a -> a -> Bool
show :: Show a => a -> String
```

look at these type signatures

```
class Eq a where
  (==) :: a -> a -> Bool
```

the Eq type class

please follow along!

```
data Weather = Sunny | Cloudy
```

defining an instance of Eq

please follow along!

```
data Weather = Sunny | Cloudy

instance Eq Weather where
```

defining an instance of Eq

please follow along!

```haskell
data Weather = Sunny | Cloudy

instance Eq Weather where
  (==) :: Weather -> Weather -> Bool
```

defining an instance of Eq

please follow along!

```
data Weather = Sunny | Cloudy

instance Eq Weather where
  (==) :: Weather -> Weather -> Bool
  Sunny == Sunny = True
  Cloudy == Cloudy = True
  _ == _ = False
```

defining an instance of Eq

please follow along!

```haskell
data Foo = A Int | B Weather

instance Eq Foo where
  (==) :: Foo -> Foo -> Bool
```

please follow along!

```haskell
data Foo = A Int | B Weather

instance Eq Foo where
  (==) :: Foo -> Foo -> Bool
  (A i1) == (A i2) = i1 == i2
  (B w1) == (B w2) = w1 == w2
  _ == _ = False
```

please follow along!

```
data Foo = A Int | B Weather

instance Eq Foo where
  (==) :: Foo -> Foo -> Bool
  (A i1) == (A i2) = i1 == i2
  (B w1) == (B w2) = w1 == w2
  _ == _ = False
```

please follow along!

```
data Foo = A Int | B Weather

instance Eq Foo where
  (==) :: Foo -> Foo -> Bool
  (A i1) == (A i2) = i1 == i2  Int -> Int -> Bool
  (B w1) == (B w2) = w1 == w2
  _ == _ = False
```

please follow along!

```haskell
data Foo = A Int | B Weather

instance Eq Foo where
  (==) :: Foo -> Foo -> Bool
  (A i1) == (A i2) = i1 == i2
  (B w1) == (B w2) = w1 == w2  Weather -> Weather -> Bool
  _ == _ = False
```

please follow along!

```
data Foo = A Int | B Weather

instance Eq Foo where
  (==) :: Foo -> Foo -> Bool
  (A i1) == (A i2) = i1 == i2
  (B w1) == (B w2) = w1 == w2
  _ == _ = False
```

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
```

the full Eq type class

```haskell
class Eq a where
  (==) :: a -> a -> Bool
  x == y = not (x /= y)

  (/=) :: a -> a -> Bool
  x /= y = not (x == y)
```

default implementations

(exercise: Eq instance for tree)

(example: Eq instance for polymorphic tree)

type class constraint

```haskell
elem :: Eq a => a -> [a] -> Bool
elem _ [] = False
elem e (x : xs) = e == x || elem e xs
```

type class polymorphic functions

```haskell
elem :: Eq a => a -> [a] -> Bool
elem e = any (== e)
```

type class polymorphic functions

(exercise: removeAll)

```haskell
lookup :: Eq a => a -> [(a, b)] -> Maybe b
```

type class polymorphic functions

type class declaration

```
class Eq a where
  (==) :: a -> a -> Bool
```

type class instance declaration

```
instance Eq Foo where
  (==) :: Foo -> Foo -> Bool
  (A i1) == (A i2) = i1 == i2
  (B w1) == (B w2) = w1 == w2
  _ == _ = False
```

function with type class constraint

```
elem :: Eq a => a -> [a] -> Bool
elem e = any (== e)
```

```
class Show a where
  show :: a -> String
```

the Show type class

```
instance Show Weather where
  show :: Weather -> String
  show Sunny = “Sunny”
  show Cloudy = “Cloudy”
```

defining an instance of Show

```
instance Show Foo where
  show :: Foo -> String
  show (A i) = “A” ++ show i
  show (B w) = “B” ++ show w
```

defining an instance of Show

```
data Foo = A Int | B Weather
 deriving (Eq, Show)
```

deriving

(exercise: duration)

(homework overview)