Type Classes + Quickcheck

CSE 130
11.19.21
Why do we need type classes?

add_int :: Int -> Int -> Int
add_double :: Double -> Double -> Double
Why do we need type classes?

\[
\text{add\_int} :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}
\]

\[
\text{add\_double} :: \text{Double} \rightarrow \text{Double} \rightarrow \text{Double}
\]

Does polymorphism solve the problem?
Why do we need type classes?

add_int :: Int \rightarrow Int \rightarrow Int
add_double :: Double \rightarrow Double \rightarrow Double

Does polymorphism solve the problem?

add :: a \rightarrow a \rightarrow a

Too general!
Why do we need type classes?

add_int :: Int -> Int -> Int
add_double :: Double -> Double -> Double

Does polymorphism solve the problem?

add :: Num a => a -> a -> a
Examples

GHC.List

elem :: Eq a => a -> [a] -> Bool
maximum :: Ord a => [a] -> a

Data.Set

insert :: Ord a => a -> Set a -> Set a
Define a type class

```haskell
class TypeName a where
    fun1 :: a -> a
    fun2 :: a -> String
    fun3 :: a -> a -> Bool
```
Define a type class

Name of the type class

```
class TypeName a where
    fun1 :: a -> a
    fun2 :: a -> String
    fun3 :: a -> a -> Bool
```
Define a type class

class TypeName a where

  fun1 :: a -> a
  fun2 :: a -> String
  fun3 :: a -> a -> Bool

Name of the type class

Type variable as a placeholder for the specific type that will implement this class
Define a type class

class TypeName a where

   fun1 :: a -> a
   fun2 :: a -> String
   fun3 :: a -> a -> Bool
Define a type class

class TypeName a where

fun1 :: a -> a

fun2 :: a -> String

fun3 :: a -> a -> Bool
Define a type class

class Num a where

(+) :: a -> a -> a

(-) :: a -> a -> a

(*) :: a -> a -> a
Define a type class

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

class Eq a => Ord a where
  compare :: a -> a -> Ordering
  (<) :: a -> a -> Bool
  (<=) :: a -> a -> Bool
  (>) :: a -> a -> Bool
  (>=) :: a -> a -> Bool
```
Define a type class

class Eq a where

(==) :: a -> a -> Bool
( /=) :: a -> a -> Bool

class Eq a => Ord a where

compare :: a -> a -> Ordering
(<) :: a -> a -> Bool
(<=) :: a -> a -> Bool
(>) :: a -> a -> Bool
(>=) :: a -> a -> Bool
How to use a type class

Color = Red | Blue | Green
Approach #1: deriving

Color = Red | Blue | Green    deriving (Eq, Ord, Show)
Approach #2: defining instances

Color = Red | Blue | Green

instance Eq Color where
   (==) Red Red = True
   (==) Blue Blue = True
   (==) Green Green = True
   (==) _ _ = False
Practice
Property based testing with Quickcheck

Writing tests sucks!
A different approach

Property based testing:

Randomly generate inputs to your function, and check that certain properties hold
A different approach

Property based testing:

Randomly generate inputs to your function, check that certain properties hold for all inputs

Property = A boolean-valued function
For example

prop :: [a] -> Bool
prop xs = reverse (reverse xs) == xs
Practice