Notes on Type Classes

* Interface that defines some behavior
  + Supports and implements
  + Like java interfaces, but better
* Used in class constraints
  + Basically, put constraints on type declarations/variables
* Examples:
  + Eq -> define equality testing
  + Ord -> define ordering (for comparisons)
* Nothing to do with classes
* When implementing a typeclass, one is simply adding a series of functions that a typeclass can add with a type that belongs to it
  + Ex: Eq provides (==) and (/=)
    - It also provides that (==) is the same as not (/=) and vice versa
  + If needed to be added to a new type, the typeclass could be derived and the behavior for those two functions could be defined
* When using Maybe a, things get interesting
  + Must surround in parentheses to use as one variable
  + May need to use class constraints on types within the maybe
    - Class constraints in class declarations are used to make subclasses of Typeclasses typically
    - Ex: Making the contents of Maybe also be a part of the Eq typeclass
* Instances are ways of overloading typeclass operations to give them specific properties
* Typeclasses can inherit from other typeclasses by applying a class constraint in the class definition
  + Ex: Ord would want to inherit from Eq, but provide a little more functionality (comparisons and min/max)
* Kinds – What are they?
  + A second type of type system in Haskell
  + Basically, tell the compiler when there is some sort of error in how a data type is being constructed
    - \* means 1 concrete type
      * Ex: Int, Char, Bool, etc
    - \* -> \* means takes a concrete type and returns a concrete type
      * Ex: Maybe a, Tree a, [a]
    - Etc.
  + Functors must all have a kind of type \* -> \*