Declaring types and classes

type declarations can also have parameters

EX:

The parameter in this example is a

type Pair a = (a,a)

mult :: Pair Int -> Int

mult (m,n) = m\*n

copy :: a -> Pair a

copy x = (x,x)

Can also nest type delcarations

type Pos = (Int, Int)

type Trans = Pos -> Pos

Cannot be recursive

* type Tree = (Int, [Tree])

Data declaration

* a new type can be defined by specifying its values using a data declaration
* the values are called the constructors
* the type and constructor names must be capitalised

Ex:

* data Bool = False | True

constructors in a data declaration can also have arguments

EX:

* data Shape = Circle Float
* | Rect Float Float

Pattern matching for example can be used to specify

* area :: Shape -> Float
* area (Circle r) = pi \*r^2
* area (Rect x y) = x \* y

Data declarations can also have parameters

EX:

* data maybe a = Nothing | Just a

can use maybe so for example maybe Bool (options are Nothing, Just True or Just False)

Then use the nothing as the exception case instead of throwing an error or using a default value

NewType declaration

* more efficient than using a datatype structure
* single constructor
* single argument
* Ex:
  + newtype Nat = N int

Recursive Types

* can define types In terms of themselves
* data Nat = Zero | Succ Nat
  + this means that a value of type Nat, is either of the value Zero or of the form Succ n, where n :: Nat
  + Nat contains the following infinite sequence of values, Zero, Succ Zero, Succ (Succ Zero) …
  + Where Succ represents the successor function +1 and Zero represents 0

Type Classes

Use the class keyword to declare a new class

EX:

class Eq a where

(==), (≠) :: a -> a -> Bool

x ≠ y = not (x ==y)

This means that every type that belongs to this class needs to support the 2 operators == and ≠

Take 2 elements of the type and return a Bool

It then says that the operator ≠ is equal to not (x==y)

This means that once a definition is provided for x==y then whenever this definition is false the operator ≠ is used

Classes then need to be instantiated by providing implementations for corresponding operators

Instance Eq Bool where

False == False = True

True == True = True

\_ == \_ = False

Because bool is an instance of the class Eq it implements this class every time it is used

Extending a class

* Inherits all the operators from the parent class
* An example is Ord can inherit from the Eq class