**type classes**

is this an element of this list (of type, say, `Bool`)?

{\[
\text{isinBList} :: \text{Bool} \rightarrow [\text{Bool}] \rightarrow \text{Bool}
\]
\[
isinBList \ x \ [ ] = \ \text{False}
\]
\[
isinBList \ x \ (y : ys) = (x == \text{Bool} \ y) || \text{isinBList} \ x \ ys
\]

if the list was of type `[Int]`

{\[
\text{isinIList} :: \text{Int} \rightarrow [\text{Int}] \rightarrow \text{Bool}
\]
\[
isinIList \ x \ [ ] = \ \text{False}
\]
\[
isinIList \ x \ (y : ys) = (x == \text{Int} \ y) || \text{isinIList} \ x \ ys
\]

generically

{\[
\text{isinList} :: a \rightarrow [a] \rightarrow \text{Bool}
\]

and restrict `a` to only those types that have equality defined over them
**overloading**

There are two kinds of function which work over more than one class:

- **polymorphic** - single definition which works over all its types
  
  \[
  \text{length} :: [\text{a}] \rightarrow \text{Int} \\
  \text{length} \ [ \] = 0 \\
  \text{length} \ (x : \text{xs}) = 1 + \text{length} \ \text{xs}
  \]

- **overloaded** - (e.g. equality, +, show) that can be used for many types but have different definitions for different types

**type classes** - collection of types

**equality type class (Eq)**

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

```
instance Eq Int
instance Eq Float
instance Eq Bool
instance Eq Char
instance Eq [Int, Bool]
instance Eq [Char]
```
same3 :: Int -> Int -> Int -> Bool
same3 m n p = (m == n) && (n == p)

\[ \text{same3} \quad \text{in the context of} \quad \text{Eq} \quad a \Rightarrow a \rightarrow a \rightarrow Bool \]

\[ \text{same3} m n p = (m == n) && (n == p) \]

\[ \text{thus restricting a to types such as:} \]

\[ \text{• Char,} \]

\[ \text{• Int,} \]

\[ \text{• (Int, Bool),} \]

\[ \text{• Float,} \]

\[ \text{etc.} \]

isinList :: Eq a => a -> [a] -> Bool
isinList x [] = False
isinList x (y : ys) = (x == y) || isinList x ys

\[ \text{isinList} \quad \text{a -} \quad \text{• Bool} \]

\[ \text{• Char} \]

\[ \text{• Int} \]

\[ \text{• (Int, Int)} \]

---

\[ \text{class Eq a where} \]

\[ (==), (/=) :: a \rightarrow a \rightarrow Bool \]

\[ x /= y = \text{not} (x == y) \]

\[ x == y = \text{not} (x /= y) \]

definition of Eq

\[ \text{signature} \]

\[ \text{derived class Ord} \]

\[ \text{class Eq a => Ord where} \]

\[ (<), (<=), (>, ) :: a \rightarrow a \rightarrow Bool \]

\[ \text{max, min} :: a \rightarrow a \rightarrow a \]

\[ \text{compare} :: \text{Ordering} \]

\[ \text{compare x y} \]

\[ | x == y = \text{EQ} \]

\[ | x <= y = \text{LT} \]

\[ | \text{otherwise} = \text{GT} \]

\[ \text{class Ord inherits the operations of Eq} \]
**class Enum**

```
class Ord a => Enum a where
  toEnum :: Int -> a
  fromEnum :: a -> Int
  enumFrom :: a -> [a]
  enumFromThen :: a -> a -> [a]
  enumFromTo :: a -> a -> [a]
  enumFromThenTo :: a -> a -> a -> [a]
```

`fromEnum` and `toEnum` convert between `a` and `Int`.

In case of `Char`,

```
ord :: Char -> Int
ord = fromEnum
```

most types belong to `Show`

```
class Bounded a where
  minBound, maxBound :: a

type ShowS = String -> String

class Show a where
  showPrec :: Int -> a -> ShowS
  show :: a -> String
  showList :: [a] -> ShowS
```

types

- `Int`, `Char`, `Bool`, `Ordering`
Numeric types in Haskell:

- **Int**: fixed precision integers
- **Integer**: all integers represented accurately
- **Float**: floating point numbers
- **Double**: float in double precision
- **Rational**:

The basic class to which all numeric types belong is **Num**.

```haskell
class (Eq a, Show a) a => Num a where
  (+), (-), (*) :: a -> a -> a
  negate        :: a -> a
  abs, signum  :: a -> a
  fromInteger  :: Integer -> a
  fromInt      :: Int -> a

  x - y = x + negate y
  fromInt = fromIntegral
```

Integer types belong to the class **Integral** whose signature include:

- **quot**, **rem** :: a -> a -> a
- **div**, **mod** :: a -> a -> a