# type classes

So, what is the type of 42?

```
42 :: Num a => a
```

What does this mean?
`Num` is a type class.

`Num a` is a type class constraint.

`Num a => a` means some type `a` in the class `Num`.

And what is the `Num` class? All types in this class must implement addition, subtraction, multiplication, negation, absolute value, and other. See Haskell documentation for details.

# type classes

Type classes offer a controlled approach to overloading.

There are a number of predefined type classes: `Eq`, `Ord`, `Show`, `Read`, `Num`, and more.

You can create instances of these classes.

You can also create your own classes and instantiate them.

(These are NOT like Python/Java classes. More like Java interfaces.)

# type classes

The Eq type class is all types with equality defined.

Types in this class provide == and /=.

```
member _ [] = False
member y (x : xs) = x == y || member y xs
```

```
prompt> :type member

member :: (Eq t) => t -> [t] -> Bool
```

(Eq t) is a type class constraint.

All the base types (Int, Bool, etc.) are members of Eq.

Let's look more closely at what type classes are.

# type classes

A simple way to create a member of `Eq`:

```
data Btree a = Empty | Node (Btree a) a (Btree a)
               deriving Eq
```

```
prompt> (Node Empty 4 Empty) == (Node Empty 5 Empty)
False

prompt> (Node Empty 4 Empty) /= (Node Empty 5 Empty)
True

prompt> (Node Empty 4 Empty) == (Node Empty 4 Empty)
True
```

## type classes

We may wish to provide a non-derived equality method.

```
data First = Pair Int Int

instance Eq First where
    (Pair x _) == (Pair y _) = (x == y)

prompt> (Pair 1 3) == (Pair 2 3)

False

prompt> (Pair 1 3) == (Pair 1 4)

True
```

# type classes

More general:

```
data First a = Pair a a


instance Eq a => Eq (First a) where
    (Pair x _) == (Pair y _) = (x == y)

prompt> (Pair [1, 2, 3] [1]) == (Pair [1, 2, 3] [3])
True
prompt> (Pair [1] [2]) == (Pair [2] [2])
False
```

# type classes

Providing a **default definition** of a function in the type class definition:

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

Either of these default definitions may be overridden.

```
prompt> (Pair [1, 2, 3] [1]) /= (Pair [1, 2, 3] [3])
False
prompt> (Pair [1] [2]) /= (Pair [2] [2])
True
```

# type classes

`Ord` inherits from `Eq` and specifies the four comparison operators `<`, `<=`, `>`, `>=`. It gives default definitions for `min` and `max` in terms of these.

There is also a three-way `compare` function which returns `LT`, `EQ`, and `GT`.

Most basic datatypes are instances of `Ord`, and user-defined datatypes can derive `Ord` (lexicographic ordering).

# type classes

Show specifies the method show :: a -> String.

Read specifies the method read :: String -> a and can be used for parsing.

Num inherits from Eq, and specifies +, -, *, negate, abs, and signum.

Division in handled by Integral and Fractional, which inherit from Num.

Use :info <typeclass> too see the instances of a typeclass.

# type classes

Defining our own type class:

```
class YesNo a where
    yesno :: a -> Bool
```

and some instances:

```
instance YesNo Integer where
    yesno 0 = False
    yesno _ = True

instance YesNo [a] where
    yesno [] = False
    yesno _ = True
```

# type classes

```
instance YesNo Bool where
    yesno x = x

instance YesNo (BTree a) where
    yesno Empty = False
    yesno _ = True
```

Then

```
prompt> yesno []

False
prompt> yesno ""

False
prompt> yesno [1, 2, 3]

True
prompt> yesno "abc"

True
```

# type classes

cont.

```
prompt> yesno Empty
False
prompt> yesno (Node "a" Empty Empty)
True
prompt> yesno True
True
prompt> yesno (1 == 0)
False
prompt> yesno 0
False
prompt> yesno 42
True
```

# type classes

cont.

```
prompt> :t yesno
yesno :: (YesNo a) => a -> Bool

prompt> :info YesNo
class YesNo a where yesno :: a -> Bool
   -- Defined at /...path.../filename.hs:140:6-10
instance YesNo Integer
  -- Defined at /...path.../filename.hs:(143,0)-(145,17)
instance YesNo [a]
  -- Defined at /...path.../filename.hs:(147,0)-(149,17)
instance YesNo Bool
  -- Defined at /...path.../filename.hs:(151,0)-(152,19)
instance YesNo (BTree a)
  -- Defined at /...path.../filename.hs:(154,0)-(156,17)
```