Definition : Type constructors

Used only in type level, type signatures, and typeclass declarations and instances. Types are static and resolve at compile time.

Definition : Data constructors

Data constructors construct the values at term level, values you can interact with at runtime.

Kinds are types of types, 1 level up, and is represented with *.

1 Type and Data

Types are static and resolve at compile time. Data are what we're working at runtime.

Compile time is when the program is getting compiled by GHC before execution. Runtime is the actual execution.

2 Data constructor arities

Arity refers to the number of arguments of a function or constructor takes. *Nullary* is a function taking no arguments (null - ary). *Unary* refers to data constructors that take one argument. *Products* are those take more than one arguments.

Definition : Algebraic

Algebraic datatypes are algebraic since the patterns of argument structures using 2 basic operations: *sum* and *product*.

Definition : Cardinality

The cardinality of a datatype is the number of possible values it defines.

The value will be constructed at runtime from the argument we applied it to.

newtype

A newtype has no runtime overhead, as it reuses the representation of the type it contains.

```haskell
{-# LANGUAGE GeneralizedNewtypeDeriving #-}

class TooMany a where
  tooMany :: a -> Bool

instance TooMany Int where
  tooMany n = n > 42

newtype Goats =
  Goats Int deriving (Eq, Show, TooMany)
```

With this pragma, no need to declare again.

3 Record syntax

Records are product types with additional syntax to provide convinient accessors to fields within the record.

```haskell
data Person =
  Person { name :: String
         , age :: Int }
         deriving (Eq, Show)
```

They are just functions that take a member of the product. (age papu...)

And records are just syntax to create field references.

Distributive property

Just like $a.(b + c) = a.b + a.c$, that's the normal form.

Constructing and deconstructing values (duality)

Definition : Type synonyms

Try to avoid using *type synonyms* with unstructured data like text or binary. It's best used when you want something *lighter weight* than newtypes but also want the type signatures to be more explicit.

The total values is the product of the number of inhabitants.

Deconstructing values

Recall that *catamorphism* are means of deconstructing data, breaking down any datatype. If the spine of a list is the structure of it, then fold can reduce the structure.

Unpacking or deconstructing the contents of a product type.

Function type is exponential

The number of inhabitants of $a -> b$ is b^a.

4 Higher-kinded datatypes

Recall that kinds are types of type constructors. Kinds are not type until they are fully applied.

Lists are polymorphic until the type's argument has been fully applied:

```haskell
data []   a   =   [] | a   :   [a]
--    [1]  [2]     [3]   [4]  [5]  [6]
```

"[]" : Type constructor for list (special syntax), take "a" as argument. The colon is an infix data constructor, it is the product of a and [a].

Infix type and data constructors

An operator that has non-alphanumeric name is **infix** by default. Any operator that starts with (:) must be an infix type or data constructor. All infix data constructors start with a colon. The type constructor of functions, ($\rightarrow$), is the only exception. (:: is type assertions)

Definition : The difference

Type constructors are functions one level up, structuring things that cannot exist at runtime - it's purely static and describes the structure of your types.

5 Binary Tree

```
foldTree :: (a -> b -> b) -> b -> BinaryTree a -> b
foldTree f b Leaf = b
foldTree f b (Node left a right)
  = foldTree f (f a (foldTree f b left)) right


--           f _____________b___________ BinaryTree
```

This is, roughly, what is being generated by the automatic *deriving Foldable*.

Magic spell: {# LANGUAGE DeriveFoldable #}, and then foldTree = foldr.

The above is a *sequential* fold, which first folds over the left part, then over the middle element, then over the right.

6 Hutton's Razor