Learn You a Physics for Great Good!

>>> WORK IN PROGRESS <<<

Dimensions / Value-level dimensions

Value-level dimensions

```
module Dimensions.ValueLevel
    ( Dim(..)
    , mul
    , div
    , length
    , mass
    , time
    , current
    , temperature
    , substance
    , luminosity
    , one
    ) where

import Prelude hiding (length, div)
```

From the introduction, two things became apparanent:

1. Given the unit of a quantity, its dimension is known implicitly.
2. If we only work with SI-units, there is a one-to-one correspondence between dimensions and units.

We'll use these facts when implementing dimensions. More precisely, "length" and "metre" will be interchangable, and so on.

A dimension can be seen as a product of the base dimensions, with an individual exponent on each base dimension. Because the 7 base dimensions are known in advance, we can design our data type using this fact.

```
data Dim = Dim Integer -- Length
               Integer -- Mass
               Integer -- Time
               Integer -- Current
               Integer -- Temperature
               Integer -- Substance
               Integer -- Luminosity
  deriving (Eq)
```

Each field denotes the exponent for the corresponding base dimension. If the exponent is `0`, the base dimension is not part of the dimension. Some examples should clarify.

```
length      = Dim 1 0 0 0 0 0 0
mass        = Dim 0 1 0 0 0 0 0
time        = Dim 0 0 1 0 0 0 0
current     = Dim 0 0 0 1 0 0 0
temperature = Dim 0 0 0 0 1 0 0
substance   = Dim 0 0 0 0 0 1 0
luminosity  = Dim 0 0 0 0 0 0 1


velocity     = Dim 1 0 (-1) 0 0 0 0
```

Velocity is m/s or equivalently $m^1 * s^{-1}$. This explains why the exponents are as above.

Noticed how we used "m" (for metre) for implicitly refering to the dimension "length"? It's quite natural to work this way.

Exercise Create values for acceleration, area and charge.

▼ Solution

```
acceleration = Dim 1 0 (-2) 0 0 0 0
area         = Dim 2 0 0    0 0 0 0
charge       = Dim 0 0 1    1 0 0 0
```

Multiplication and division

Dimensions can be multiplied and divided. Velocity is, as we just saw, a division between length and time. Multiplication and division of dimensions are performed as if they were regular numbers, or variables holding numbers, and hence they follow the power laws. That is, to multiply, the exponents of the two numbers are added, and to divide, the exponents are subtracted.

```
mul :: Dim -> Dim -> Dim
(Dim le1 ma1 ti1 cu1 te1 su1 lu1) `mul` (Dim le2 ma2 ti2 cu2 te2 su2 lu2) =
  Dim (le1+le2) (ma1+ma2) (ti1+ti2) (cu1+cu2) (te1+te2) (su1+su2) (lu1+lu2)
```

Exercise Implement a function for dividing two dimensions.

▼ **Solution**

```
div :: Dim -> Dim -> Dim
(Dim le1 ma1 ti1 cu1 te1 su1 lu1) `div` (Dim le2 ma2 ti2 cu2 te2 su2 lu2) =
  Dim (le1-le2) (ma1-ma2) (ti1-ti2) (cu1-cu2) (te1-te2) (su1-su2) (lu1-lu2)
```

It's now possible to construct dimensions in the following fashion.

```
velocity' = length `div` time
area'     = length `mul` length
force     = mass   `mul` acceleration
momentum  = force  `mul` time
```

A dimension we so far haven't mentioned is the *scalar*, which shows up when working with, for example, coefficients of friction. It's dimensionless because it arises from division of two equal dimensions. The case of coefficients of friction looks like

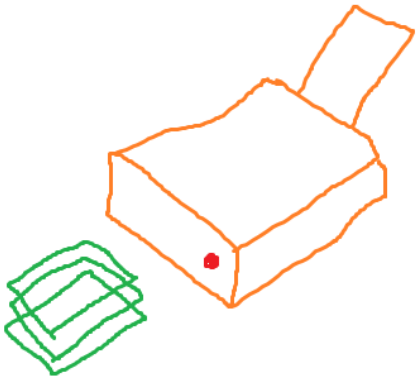
$$F_{friction} = \mu * F_{normal} \iff \mu = \frac{F_{friction}}{F_{normal}}$$

Exercise Create two values, which represent the scalar. They should of course have the same value, but be created in two different ways. One by writing the exponents explicitly. One by dividing two equal dimensions.

▼ Solution

```
one  = Dim 0 0 0 0 0 0 0
one' = force `div` force
```

Pretty-printer



The purpose of value-level dimensions is to be able to print 'em nicely. The pretty printer should be function with the type

```
showDim :: Dim -> String
```

meaning it shows a dimension as a string. But how to actually implement this is not the interesting part of this tutorial. Hence we skip it.

We use `showDim` to make `Dim` an instance of `Show`

```
instance Show Dim where
  show = showDim
```

Now dimensions are printed in a quite pretty way in GHCi

```
ghci> momentum
kg*m/s
```

Note that the SI-unit of the dimensions is used for printing.

The result from this section is the ability to multiply, divide and print dimensions. The next step is to test these operations and see if they actually work.