Week 2

Enumeration Types

Enumeration in Haskell. The following declares a new type called **'Foo'** with five data constructors, the (only) values of type Foo. The *'Deriving Show'* tells GHC to automatically gneerate the default code for converting Foo's to Strings.

```
data Foo = One
         | Two
         | Three
         | Four
         | Five
    deriving Show

-- This is how it will convert

One :: Foo
one = One
```

Using Enumerations

Enums can be used in a variety of ways, for example

```
data Person = Person String Int Foo
    deriving Show

--Store a person's name, age, and foo

chris :: Person
chris = Person "Chris" 20 Three
```

Case Expressions

Case expressions look like

```
case expression of
    pat1 -> ex1
    pat2 -> exp2
    ...
```

The expression is matched against the patterns. The patterns are matched in order. The first matched pattern is chosen.

```
foo = case "Hello" of
        []      -> 3
        ('H':s) -> length s
        _       -> 7
```

Recursive Data Types

Data types can be recursive, defined in terms of themselves. We can define lists recursively: *A list is either empty, or a single element followed by a list*

```
data IntList = Empty | Cons Int IntList
```