COMP105 Lecture 19

Custom Types

The type keyword

The **type** keyword gives a new name to an existing type

- All types must start with capital letters

```
type String' = [Char]
```

```
exclaim :: String' -> String'
exclaim str = str ++ "!"
```

```
ghci> exclaim "hello"
"hello!"
```

The type keyword

type is useful when you want to give a **meaningful name** to a complex type

```
type VoteResults = [(Int, String)]
```

```
results :: VoteResults
results = [(2, "red"), (1, "blue"), (1, "green")]
```

```
ghci> head results
(2,"red")
```

The `data` keyword

The **data** keyword is used to create an entirely new type

```
data Bool' = True | False
```

- | should be read as “or”
- each of the values is a **constructor**

The data keyword

```
data Direction = North | South | East | West
```

```
rotate North = East
rotate East = South
rotate South = West
rotate West = North
```

```
ghci> :t rotate
rotate :: Direction -> Direction
```

Type classes

By default, a new data type is **not** part of any type class

```
ghci> rotate North

No instance for (Show Direction) arising from ...
```

Type classes

We can use the **deriving** keyword to fix this

```
data Direction = North | South | East | West
                                      deriving (Show)
```

```
ghci> rotate North
East
```

Haskell automatically writes the `show` function for us

- You can override this if you want

Type classes

Haskell can **automatically** implement the following type classes

- `Show` – will print out the type as it is in the code
- `Read` – will parse the type as it is in the code
- `Eq` – the natural definition of equality
- `Ord` – constructors that come first are smaller

Exercises

1. Use the `type` keyword to create a type called `Marks` that is a pair where the first element is a string (for a name), and the second element is a list of integers (giving marks)

2. Write a custom type `Color` with three values `Red`, `Blue`, and `Green`. Make this type an instance of `Show` and `Read`

3. Create `toRGB :: Color -> (Float, Float, Float)` that turns a color into its RGB value, where red maps to $(1, 0, 0)$, green maps to $(0, 1, 0)$, and blue maps to $(0, 0, 1)$