型を自分で作ろう

kazu.hs

- 3までしか数えられない数を作ってみましょう。
- 次のように入力すると、新しい型Countが作れます

```
> data Count = One | Two | Three | Huh
    deriving (Eq, Show)
```

- Count型の値は、One, Two, Three, Huhのいづれかです。（ひとつ、ふたつ、みっつ、ハァわからん）
- インタプリタに入力する場合に全体を1行で入力して下さい。

型を自分で作ろう

- 次のように入力すると、新しい型Countが作れます。

  data　型　＝　値　｜値　｜値　　｜値

```
> data Count = One | Two | Three | Huh
```

Preludeの既存の型も、まったく同じ構文で定義されています。

```
> :info Bool
```

data　型　＝　値　｜値

```
data Bool = False | True  -- Defined in
‘GHC.Types’
```

Count型の値の計算をするには？ CountをNumのインスタンスにします。

```
instance Num Count where
  One + One = Two
  One + Two = Three
  Two + One = Three
  _   + _   = Huh

  One * x   = x
  x   * One = x
  _   * _   = Huh
```

関数定義などのパターンマッチにおける
アンダースコア _ 記号は「その他」の意味です。

パターンマッチにおいて x などの変数を
用いると、パターンの特定の位置にくる値
に名前をつけられます。

引き算(-)、絶対値 abs、符号数signum、整数からの変換fromInteger
といった、Numクラスの残りの関数も実装してみよう。(kazu.hsには実装済みです。)

Count型は普通のHaskellの型と同様に扱えます。

```
$ ghci kazu.hs
> :t One
One :: Count
> :t Huh
Huh :: Count
> One + Two
Three
> Two + Two
Huh
> :info Count
data Count = One | Two | Three | Huh   -- Defined at kazu.hs:1:1
instance Eq Count -- Defined at kazu.hs:2:22
instance Num Count -- Defined at kazu.hs:4:10
```

stack経由で試す場合は、

$ stack ghci xxx

と書くと、「xxxというフォルダ内のプロジェクトの環境でghciを起動せよ」という意味になってしまうので、
次のように、ghciを起動したあと `:l` (コロンえる)
でファイルを読み込んでください。

```
$ stack ghci
...
> :l "kazu.hs"
...
> Two * One - One
One
```

- 1,2,3 ... 等がCount型のリテラルとして使えますが、少し難しい計算をさせると途中でオーバーフローしてHuh?になってしまいます。
- このことから、数式全体がCount型で演算されていることがわかります。Integerで計算してからCountに変換されているのではない。

```
> One + One + 1
Three
> 4
4
> 4 :: Count
Huh
```

```
> 1 + 2 - 1 :: Count
Two
> 1 + 3 - 1 :: Count
Huh
> 3 - 1 + 1 :: Count
Three
> 1 - 2 + 3 :: Count
Huh
```

再帰的なインスタンス定義

- 型クラスのメンバ関数定義の右辺では、通常の関数定義とおなじく、任意の式をつかうことができます。

「a がNumのインスタンスなら、Maybe a もNumのインスタンスだよ！」

```
> :{
*Main| instance Num a => Num (Maybe a) where
*Main|   Just x + Just y = Just (x + y)
*Main|   _      + _      = Nothing
*Main| :}
```

型Maybe aの足し算を、型aの足し算を用いて、定義しています

```
<interactive>:17:10: Warning:
    No explicit implementation for
      ‘*’, ‘abs’, ‘signum’, ‘fromInteger’, and (either
‘negate’ or ‘-’)
    In the instance declaration for ‘Num (Maybe a)’
>
```

ところで、:{ ... :} を使うと、インタプリタに複数行を入力できます

再帰的なインスタンス定義

- Maybe Integerばかりか、Maybe (Maybe Integer)などもNumのインスタンスになりました!

```
> Just 4 + Just 9
Just 13
> Just 4 + Nothing
Nothing
> Just (Just 4) + Just (Just 1)
Just (Just 5)
```

演習問題

exercise-3-string-Num

- String型を、(＋)演算子で文字列結合できるようにしてしまいましょう。
- ヒント：型クラスNumのインスタンスにすればOK.