CHAPTER 8: RECURSIONS

Recursions

Basically, recursion is self referential composition.

```
module Factorial where

factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)

-- explaination
brokenFact1 4 =
  4 * (4 - 1)
  * ((4 - 1) - 1)
  * (((4 - 1) - 1) - 1)
  * ((((4 - 1) - 1) - 1) - 1)
```

Bottom

Bottom or $\perp$ is a term used in Haskell to refer to computations that do not successfully result in value.

```
f :: Bool -> Int
f False = 0
f _ = error $ "*** Exception: "
            ++ "Non-exhaustive"
            ++ "patterns in function f"
```

ERRORS!!!

I had a problem with **function composition** and **recursion**. Now it's pretty hard to know how it works when it comes to more compositions.

```
digits :: Int -> [Int]
digits = reverse . digits'
  where digits' n
          | n < 10    = [n]
          | otherwise = lastDigit : digits nReduced
            where divmod' = n `divMod` 10
                  lastDigit = snd divmod'
                  nReduced  = fst divmod'
```

or even this

```
digits :: Int -> [Int]
digits n =
  let y = digits' n
  in reverse y
    where digits' n
            | n < 10    = [n]
            | otherwise = lastDigit : digits nReduced
              where divmod' = n `divMod` 10
                    lastDigit = snd divmod'
                    nReduced  = fst divmod'
```

The result:

```
Prelude> digits 1234567
[6,4,2,1,3,5,7]
```