CS 5035 (Fall 2016)

### Project 5b. Higher-order functions (part 2) (first attempt by Oct 10).

Now that you understand the solution from last week:

1. Look up uncurry and divMod and explain [this solution](https://www.rosettacode.org/wiki/Luhn_test_of_credit_card_numbers#Haskell).
2. That solution assumes that the credit card number is represented as a String. Change it so that it takes an Integer as input.
3. Eliminate digitToInt and replace map (uncurry (+). (`divMod` 10)) with the approach used above, i.e., an expression that uses toDigits and concat. (Note that you have to tell GHC the type you want toDigits to produce.) In other words, write something like this.

toDigits :: String -> [Int]

toDigits = map (read . (: ""))

myLuhn :: Integer -> Bool

myLuhn = (0 ==) . (`mod` 10) . sum . concat . map (toDigits . show) .   
 zipWith (\*) (cycle [1,2]) . (toDigits . reverse . show)

This is *not* good coding style. It packs much too much into a single line of code. In your version break it into smaller pieces. For example:

functionA = toDigits . reverse . show

functionB = zipWith (\*) (cycle [1, 2])

functionC = sum . concat . map (toDigits . show)

myLuhn' = (0 ==) . (`mod` 10) . functionC . functionB . functionA

Make up meaningful names for these functions. Declare their types, understand what each does, and explain how it all fits together.

Be able to explain the types of these compositions.

* functionC . functionB . functionA. (It is Show a => a -> Int)
* functionC . functionB. (It is (Show a, Num a) => [a] -> Int)
* functionB . functionA. (It is Show a => a -> [Int])

Understand why the type of functionB . functionA is the same as the type of functionA, which is also Show a => a -> [Int].