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, consider the following, which is based on [this solution](https://www.rosettacode.org/wiki/Luhn_test_of_credit_card_numbers#Haskell). The following assumes the input values are Int not String.

1. doubleAndSum :: [Int] -> Int
2. doubleAndSum = fst .
3. foldr (\i (acc, even) -> (acc + nextStep even i, not even)) (0, False)
4. **where**
5. nextStep even i
6. | even = (uncurry (+) . (`divMod` 10) . (\*2)) i
7. | otherwise = i
8. myLuhn :: Int -> Bool
9. myLuhn = (0 ==) . (`mod` 10) . doubleAndSum . (map (read . (: ""))) . show
10. testCC :: [Bool]
11. testCC = map myLuhn [49927398716, 49927398717, 1234567812345678, 1234567812345670]
12. -- => [True, False, False, True]

The function foldr takes three arguments (check >:t foldr), but only two are given on line 3. How does that work? Would it be ok to put an extra pair of parentheses around line 3?

Be sure you can explain the type of uncurry and how uncurry (+) composes with divMod on line 6. To demonstrate your understanding write your own uncurry' and show that it works as well. Write your function as a lambda function:

uncurry' f = \<args> -> <expression>

If you want to play around with this code, you may want to define readInt and use it in place of read. This will tell Haskell that read is supposed to return an Int.

readInt :: String -> Int

readInt = read

In context Haskell figures out read’s return type from the type declaration of doubleAndSum. But if you use read out of context, it doesn’t know what type to return.

> read "3"

\*\*\* Exception: Prelude.read: no parse

> readInt "3"

3