CS 5035 (Fall 2016)

### Project 5. Higher-order functions (part 1) (first attempt by Oct 3).

Based on chapter [6 of LYH](http://learnyouahaskell.com/higher-order-functions). [Videos](https://sites.google.com/a/lclark.edu/drake/courses/pls/lesson-5-haskell-higher-order-functions). *The material in this chapter is the heart and soul of functional programming. Be sure you understand it.* Redo the credit card validation program without using recursion. Understand this solution to the credit card problem and be able to explain it. Many of these functions are expressed in “point-free” form (without parameters). Here’s a [nice video](https://www.youtube.com/watch?v=Cy7jBYr3Zvc) about point-free code.

-- digitToInt is defined in Data.Char, but we can define it ourselves

myDigitToInt :: Char -> Int

myDigitToInt = read . (:[]) -- The type tells read to produce an Int.

{-

myDigitToInt could have been defined using a parameter

myDigitToInt c = read [c]

That’s the same as

myDigitToInt c = read (c : [])

That’s the same as

myDigitToInt c = (read . (:[])) c

That’s the same as

myDigitToInt = read . (:[])

-}

toDigits :: String -> [Int]

toDigits = map myDigitToInt

cycle12 :: [Int]

cycle12 = cycle [1,2]

pairs :: [Int] -> [(Int, Int)]

pairs ds

| odd (length ds) = zip ds cycle12 -- If odd (length ds) cycle12 ends with 1

| otherwise = zip ds (tail cycle12)

doubleEveryOther :: [Int] -> [Int] -- Could use zipWith in pairs instead.

doubleEveryOther = map (\(d, m) -> d \* m) . pairs

sumDigits :: [Int] -> Int

sumDigits = sum . concat . map (toDigits . show) -- Why not just sum?

checkSum :: String -> Int

checkSum = sumDigits . doubleEveryOther . toDigits

isValid :: String -> Bool

isValid n = checkSum n `mod` 10 == 0

testCC :: [Bool]

testCC = map isValid ["79927398713", "79927398714"]

-- => [True, False]