Brian Simpson

Professor Vu

COP 4020

02/05/21

HW#2

1. Give three possible definitions for the logical or operator (||) using pattern matching.

    - **logicalOr :: Bool -> Bool -> Bool**

      True `logicalOr` True = True

      True `logicalOr` False = True

      False `logicalOr` True = True

      False `logicalOr` False = False

    - **logicalOr :: Bool -> Bool -> Bool**

      False `logicalOr` False = False

      _ `logicalOr` _ = True

    - **logicalOr :: Bool -> Bool -> Bool**

      True `logicalOr` b = True

      False `logicalOr` b = b

2. Redefine the following version of (&&) using conditionals rather than patterns:

   True && True = True

   _ && _ = False

   logicalAnd :: Bool -> Bool -> Bool

   logicalAnd a b = if a == True then b else False

3. A triple (x,y,z) of positive integers is called pythagorean if $x^2 + y^2 = z^2$. Using a list comprehension, define a function

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

   that maps an integer n to all such triples with components in [1..n]. For example:

   >pyths 5

   [(3,4,5), (4,3,5)]

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

   pyths n = [(x,y,n) | x <- [1..n], y <- [1..n], z <- [n], x^2+y^2==z^2]

4. A positive integer is perfect if it equals the sum of all of its factors, excluding the number itself. Using a list comprehension, define a function

   perfects :: Int -> [Int]

   that returns the list of all perfect numbers up to a given limit. For example:

   >perfects 500

   [6, 28, 496]

   perfects :: Int -> [Int]

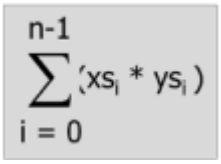
   factors_sum :: Int -> Int

   factors n = sum [x | x <- [1..n-1], n `mod` x == 0]

   perfects n = [x | x <- [1..n], factors_sum x == x]

   I opted to create two separate list comprehension functions that call upon each other to generate the list of perfect numbers instead of implementing dependent generators, as I found it to be the easier solution.

5. The scalar product of two lists of integers xs and ys of length n is given by the sum of the products of the corresponding integers:



Using a list comprehension, define a function that returns the scalar product of two lists.

scalar :: Int -> Int

scalar n = sum [x^2 | x <- [1..n]]

Assumed that the values contained within the lists of integers would be incremental, therefore an input of 3 generates two lists of length 3 that contain (1,2,3)