DM580 – EXERCISE SHEET #3

(1) Using a list comprehension, write a function `sumofsquares :: Int -> Int` such that `sumofsquares n` computes the sum $1^2 + 2^2 + \cdots + n^2$.

(2) Write a function `lengths :: [String] -> [Int]` that computes the lengths of all strings in a list, e.g., `lengths ["foo", "bar", "foobar"] == [3,3,6]`.

(3) Write a function `replicate' :: Int -> a -> [a]` that uses a list comprehension to make a list of the given value replicated the given number of times, e.g., `replicate' 3 True == [True,True,True]`.

(4) Write a function `vowels :: String -> String` that uses a list comprehension to return all vowels (in order) in a given string.

(5) Write a function `twovowels :: [String] -> [String]` that uses a list comprehension to find all strings in a list that contain exactly two vowels.

(6) Write a function `removevowels :: [String] -> [String]` that uses a list comprehension to return the strings in a given list with all vowels removed.

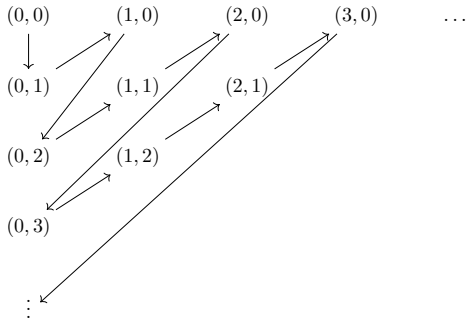
(7) A triple (x, y, z) of positive integers is *Pythagorean* iff $x^2+y^2 = z^2$. Define `pyths :: Int -> [(Int,Int,Int)]` that uses a list comprehension to find all Pythagorean triples where each component is at most the given number.

(8) The scalar product of two lists of integers $xs = [x_1, x_2, \ldots, x_n]$ and $ys = [y_1, y_2, \ldots, y_n]$ of equal length is the sum $x_1y_1 + x_2y_2 + \cdots + x_ny_n$. Write a function `scalarproduct :: [Int] -> [Int] -> Int` that computes the scalar product of two lists of integers.

(9) Section 5.5 of *Programming in Haskell* shows how to write a function `encode :: Int -> String -> String` that encodes a given string using the Caesar cipher. Write a function `decodeAll :: String -> [String]` that produces all possible Caesar cipher decodings of a given string.

(10) Show how the usual map function `map :: (a -> b) -> [a] -> [b]` can be defined using a list comprehension.

(11) *Challenge:* Write a list `antidiagonal :: [(Int,Int)]` that lists all pairs of natural numbers by following the antidiagonals, as in the illustration below.



(12) *Challenge:* You have defeated a boss in your favourite RPG, and it has dropped a bunch of loot – more than you can carry. The *RPG Loot Problem* concerns how to choose the loot that maximises total value, given that you can only carry a limited amount.

An instance of the *RPG Loot Problem* is a list of type `[(String, Int, Double)]`, where the first component of an entry is a string describing the item, the second is the item's value (in number of gold pieces, assumed to be positive), and the third is the item's weight (in kilograms, likewise assumed positive).

Write a function `loot :: [(String, Int, Double)] -> Double -> [(String, Int, Double)]` that takes a list of loot and a maximum carrying capacity, and returns the subset of the loot to choose that maximises total value and whose total weight does not exceed the maximum carrying capacity.