**Higher order haskell funcs**

1)write a higher order haskell function that takes two functions and apply the func on 2 alternate elements(first function works on odd indices,second works on even indices)

applyAlternate :: (a -> b) -> (a -> b) -> [a] -> [b]

applyAlternate \_ \_ [] = []

applyAlternate f1 f2 (x:xs) = f1 x : applyAlternate f2 f1 xs

-- Apply (+1) on odd indices, and (\*2) on even indices

applyAlternate (+1) (\*2) [1,2,3,4,5,6]

-- Output: [2,4,4,8,6,12]

2) write a higher order function such that from the given list extract multiples of 10 which are greater than a given number(so input is one int and a list)

-- Manual implementation of the 'filter' function

myFilter :: (a -> Bool) -> [a] -> [a]

myFilter \_ [] = [] -- Base case: if the list is empty, return an empty list

myFilter pred (x:xs)

| pred x = x : myFilter pred xs -- If the predicate is True, include the element

| otherwise = myFilter pred xs -- Otherwise, exclude the element

-- Manual implementation of the 'mod' function

myMod :: Int -> Int -> Int

myMod x y = x - y \* (x `div` y) -- Compute the remainder of x divided by y

-- Function to extract multiples of 10 greater than a given number

extractMultiplesOf10GreaterThan :: Int -> [Int] -> [Int]

extractMultiplesOf10GreaterThan n = myFilter (\x -> x > n && myMod x 10 == 0)

-- Example usage

main :: IO ()

main = print $ extractMultiplesOf10GreaterThan 50 [10, 20, 30, 40, 50, 60, 70, 80, 90]

3) write a curried function of taking a single number adding a constant to it squaring eg:if its 1 add 5 to it and square it=36

addAndSquare :: Int -> (Int -> Int)

addAndSquare c = (\x -> (x + c) ^ 2)