Review Problems:

scale 🡪 scale a single number by a factor

**scale n f = n \* f**

scaleTuple 🡪 scale a tuple by a factor

**scaleTuple (a,b) f = (a\*f,b\*f)**

scaleList 🡪 scale a list by a factor

**scaleList [] f = []**

**scaleList (x:xs) f = x \* f : scaleList xs f**

dotProduct 🡪 Finds dot product of two tuples

**tupleDotProduct (a,b) (c,d) = a\*c+b\*d**

dotProduct 🡪 Finds dot product of two lists

**dotProduct [] [] = 0**

**dotProduct (x:xs) (y:ys) = x\*y + dotProduct xs ys**

vectorLength 🡪 Finds length of a tuple

**vectorLength (a,b) = sqrt ((square a)+(square b))**

**where square x = x \* x**

**vector3DLength (a,b,c) = sqrt ((square a)+(square b)+(square c))**

**where square x = x \* x**

Define a function doublePos xs that doubles the positive elements in a list of integers.

(Generate a list of elements of the form 2\*x, where the x:s are the positive elements from the list xs.)

**doublePos [] = []**

**doublePos (x:xs) = (if x > 0 then x\*2 else x) : doublePos xs**

Define a function spaces n which returns a string of n spaces.

**spaces 0 = []**

**spaces n = ' ' : spaces (n-1)**

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

**evenFib n f s**

**| (f+s) > n = []**

**| otherwise = if (f+s) `mod` 2 == 0 then (f+s) : evenFib n s (f+s) else evenFib n s (f+s)**

Recursive loop:

factorial n = go n 1

where

go n ret

| n > 1 = go (n - 1) (ret \* n)

| otherwise = ret

for factorial 3 we get

go 3 1

go 2 3

go 1 6

6

**evenFib n = go n 1 2**

**where**

**go n f s**

**| (f+s) > n = []**

**| otherwise = if (f+s) `mod` 2 == 0 then (f+s) : go n s (f+s) else go n s (f+s)**

A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99.

Find the largest palindrome made from the product of two 3-digit numbers.

Steps:

palin2 = [ (x\*y,x,y) | x<-[100..999], y<-[100..999] ]

palin3 = [ (x\*y,x,y) | x<-[100..999], y<-[100..999], let str=show(x\*y), let l=length(str), l==5 ]

palin3 = [ (x\*y,x,y) | x<-[100..999], y<-[100..999], let str=show(x\*y), let l=length(str), let firstTwo=take 2 str, let rev=reverse str, let lastTwo=take 2 rev, l==5 ]

palin5 = [ (x\*y,x,y) | x<-[100..999], y<-[100..999], let str=show(x\*y), let l=length(str), let firstTwo=take 2 str, let rev=reverse str, let lastTwo=take 2 rev, let firstThree=take 3 str, let lastThree=take 3 str, l==5&&firstTwo==lastTwo ]

palin6 = [ (x\*y,x,y) | x<-[100..999], y<-[100..999], let str=show(x\*y), let l=length(str), let firstTwo=take 2 str, let rev=reverse str, let lastTwo=take 2 rev, let firstThree=take 3 str, let lastThree=take 3 rev, (l==5&&firstTwo==lastTwo) ]

palin7 = [ (x\*y,x,y) | x<-[100..999], y<-[100..999], let str=show(x\*y), let l=length(str), let firstTwo=take 2 str, let rev=reverse str, let lastTwo=take 2 rev, let firstThree=take 3 str, let lastThree=take 3 rev, (l==5&&firstTwo==lastTwo)||(l==6&&firstThree==lastThree) ]

palin8 = maximum [ (x\*y,x,y) | x<-[100..999], y<-[100..999], let str=show(x\*y), let l=length(str), let firstTwo=take 2 str, let rev=reverse str, let lastTwo=take 2 rev, let firstThree=take 3 str, let lastThree=take 3 rev, (l==5&&firstTwo==lastTwo)||(l==6&&firstThree==lastThree) ]

Add 1 to all members of a list

**addOne []= []**

**addOne (x:xs) = (x+1) : addOne xs**

**addOne' lst = [ x+1 | x <- lst ]**