19CSE313

Principles of Programming Languages

Lab 6

S Abhishek

AM.EN.U4CSE19147

1 - Define a function that maps a positive integer to its list of factors

```
factor :: Int -> [Int]
factor y = [ x | x <- [1 .. y], y `mod` x == 0]
```

```
*Main> factor 200
[1,2,4,5,8,10,20,25,40,50,100,200]
*Main> factor 10
[1,2,5,10]
*Main> factor 50
[1,2,5,10,25,50]
```

2 - Define a function that returns the list of all prime numbers up to a given limit n.

```
isPrime :: Int -> Bool
isPrime n = null [i | i <- [2 .. floor (sqrt (fromIntegral n))], n `mod` i == 0]
nPrime :: Int -> [Int]
nPrime n = [x | x <- [2 .. n], isPrime x]
```

```
*Main> nPrime 10
[2,3,5,7]
*Main> nPrime 20
[2,3,5,7,11,13,17,19]
*Main> nPrime 30
[2,3,5,7,11,13,17,19,23,29]
```

3 - Define a function pair that uses the zip function for returning the list of all pairs of adjacent elements from a list.

```
pairs :: [a] -> [(a, a)]

pairs x = zip x (tail x)
```

```
*Main> pairs [1,2,3,4,5,6]
[(1,2),(2,3),(3,4),(4,5),(5,6)]
*Main> pairs [1,2,3,4,5]
[(1,2),(2,3),(3,4),(4,5)]
*Main> pairs [1,2,3]
[(1,2),(2,3)]
```

4 - Define a function sorted that decides if the elements in a list are sorted [Check using pairs function].

```
sortF :: (Ord a) => (a, a) -> Bool

sortF (x,y) = if x <= y then True else False

sorted :: (Ord a) => [a] -> Bool

sorted x = null [ i | i <- (pairs x), sortF i == False]
```

```
*Main> sorted ["Abhi", "A"]
False
*Main> sorted ["A", "B"]
True
*Main> sorted ["A", "a"]
True
*Main> sorted ["Z", "A"]
False
```

5 - Define a function positions using zip function which will return the list of all positions of a value in a list.

```
positions :: Eq a => [a] -> a -> [Int]

positions list x = [i | (i, j) <- zip [0 .. ] list, j == x]
```

```
*Main> positions [1,0,1,0,0,1,1,1,0,1] 1
[0,2,5,6,7,9]
*Main> positions [6,7,4,2,3,4,6] 6
[0,6]
*Main> positions [1,2,3,4,5,6] 0
[]
*Main> positions [1.6,7,4,2,3,4,6] 1.6
[0]
```

6 - Define a function count to get the number of times a character occurs in a String

```
count :: (Ord a) => [a] -> a -> Int

count list x = length [ i | i <- list, i == x ]
```

```
*Main> count [1,2,3,4,2,1,4,2,5,2] 2
4
*Main> count [1,1,1,1] 1
4
*Main> count [1,1,1,1,0] 1
4
*Main> count [1,1,1,1,0] 0
1
*Main> count ["Abhi", "abhi"] "Abhi"
1
*Main> count ["A", "a", "A"] "A"
2
```

7 - Consider a triple (x,y,z) of positive integers called pythagorean if x^2 + y^2 = z^2.

- Using a lst comprehension, define a function pythFunction:: Int-> [(Int, Int, Int)] which will map an integer n to all such triples with components in [1..n]

```
pythagorean :: Int -> [(Int, Int, Int)]

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

```
*Main> pythagorean 5
[(3,4,5),(4,3,5)]
*Main> pythagorean 6
[(3,4,5),(4,3,5)]
*Main> pythagorean 10
[(3,4,5),(4,3,5),(6,8,10),(8,6,10)]
```

8 - A perfect number is a positive integer which is equal to the sum of all its factors, excluding the number itself.

- Define a function perfects :: Int->Int that returns all the perfect numbers up to a given limit n.

```
fact :: Int -> [Int]

fact x = [ i | i <- [ 1 .. (x - 1) ], x `mod` i == 0]


sumOfF :: [Int] -> Int

sumOfF [] = 0

sumOfF ( x : xs ) = x + sumOfF xs
```

```
perfect :: Int -> [Int]

perfect n = [ i | i <- [ 1 .. n ], sumOfF (fact i) == i]
```

```
*Main> perfect 500
[6,28,496]
*Main> perfect 100
[6,28]
*Main> perfect 1000
[6,28,496]
```

9 - Define a function scalar to find the scalar product of list elements of two lists xs and ys of length n.

```
scalarP :: [Int] -> [Int] -> [Int]

scalarP x y = [ i * j | ( i, j ) <- zip x y ]
```

```
*Main> scalarP [1,2,3] [3,4,6]
[3,8,18]
*Main> scalarP [1,2,3] [4,5,6]
[4,10,18]
*Main> scalarP [1,2,3] [4]
[4]
*Main> scalarP [1,2,3] []
[]
```

10 - Define the function sumsq, which takes an integer n as its argument and returns the sum of the squares of the first n integers.

```
sumSQ :: Int -> Int

sumSQ n = sum [ i*i | i <- [ 1 .. n ] ]
```

```
*Main> sumSQ 10
385
*Main> sumSQ 5
55
*Main> sumSQ 1
1
*Main> sumSQ 3
14
```

11 - Convert every character in string to uppercase and remove any digits in it

```
import Data.Char

charToUp :: String -> String

charToUp string = [ toUpper i | i <- string, i `elem` [ 'a' .. 'z' ] || i `elem` [ 'A' .. 'Z' ] ]
```

```
*Main> charToUp "Abhishek1234"
"ABHISHEK"
*Main> charToUp "Bharath Pratap Nair"
"BHARATHPRATAPNAIR"
*Main> charToUp "Chinnu Ganesh"
"CHINNUGANESH"
*Main> charToUp "123"
""
*Main> charToUp ""
""
```

12 - Define a function that extracts the upper case letters only.

```
extractUpper :: String -> String

extractUpper string = [ i | i <- string, isUpper i]
```

```
*Main> extractUpper "Abhi"
"A"
*Main> extractUpper "ABhi"
"AB"
*Main> extractUpper "ABHI"
"ABHI"
*Main> extractUpper "12345"
""
*Main> extractUpper "a3x3k"
""
```

13 - Define a function that will capitalize the first letter of a String and return the entire String.

```
caps :: String -> String

caps string = toUpper (head string) : tail string
```

```
*Main> caps "abhi"
"Abhi"
*Main> caps "a"
"A"
*Main> caps "hello"
"Hello"
*Main> caps "abhishek, this is me!"
"Abhishek, this is me!"
```

14 - Define a function cpy to make a string of n characters.

```
cpy :: Char -> Int -> String

cpy c n = replicate n c
```

```
*Main> cpy 'A' 2
"AA"
*Main> cpy 'B' 3
"BBB"
*Main> cpy 'C' 4
"CCCC"
*Main> cpy 'D' 5
"DDDDD"
```

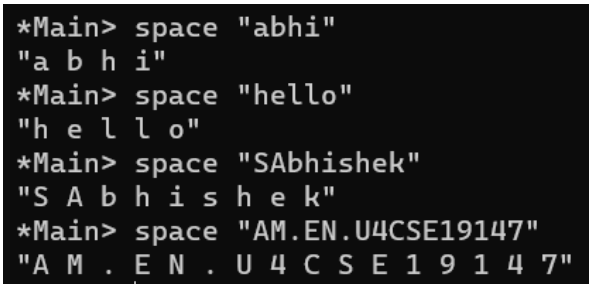
15 - Define a function to place space characters between characters in a string

```
space :: String -> String

space ( x : xs )

 | length(xs) == 0 = x : ""

 | otherwise = x : " " ++ space xs
```



Thankyou!!