LAB SHEET - 6

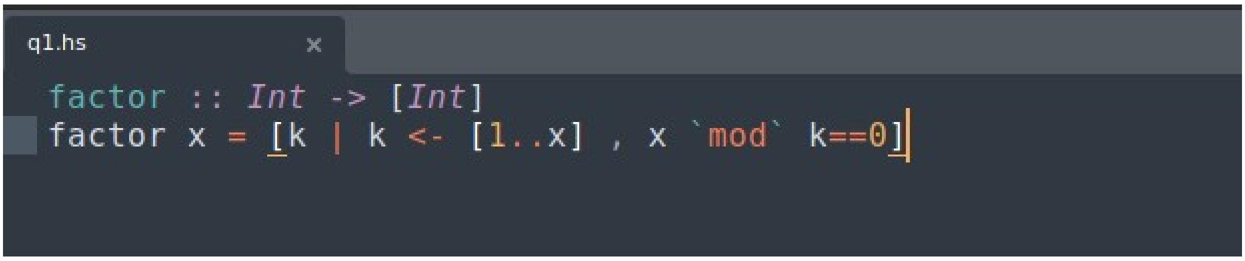
NAME : J.SAMPAT SRIVATSAV

ROLL NUMBER : AM.EN.U4CSE19125

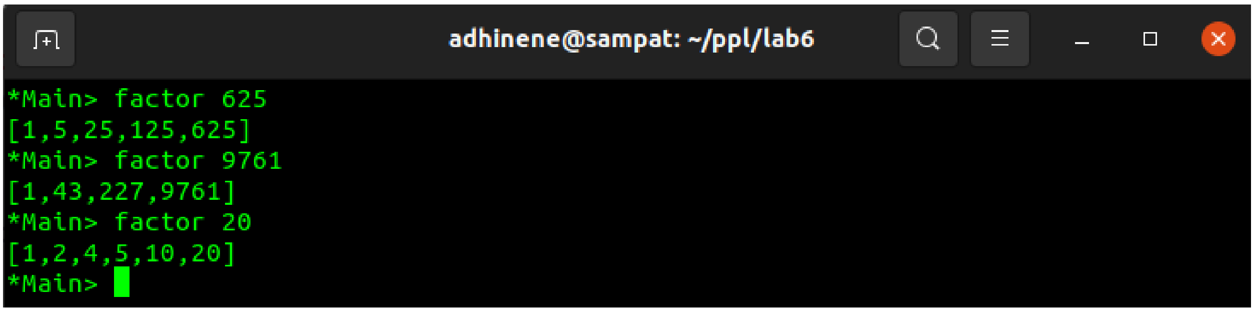
BATCH : C.S.E – B

Q1)

1. Using list comprehension, define a function that **maps a positive integer to its list of factors**.



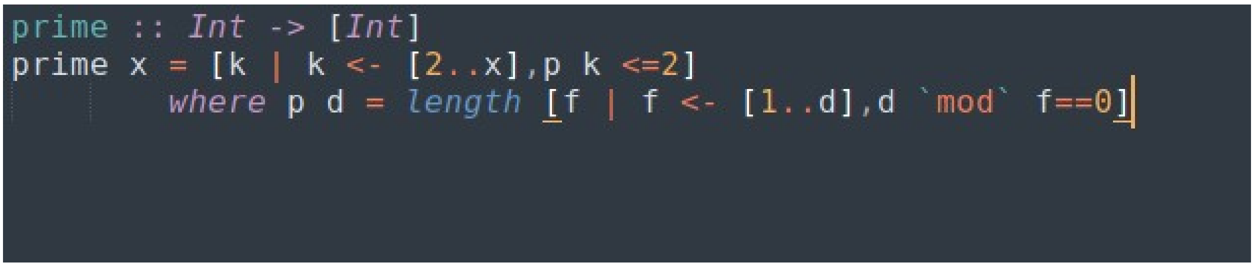
```haskell
factor :: Int -> [Int]
factor x = [k | k <- [1..x] , x `mod` k==0]
```



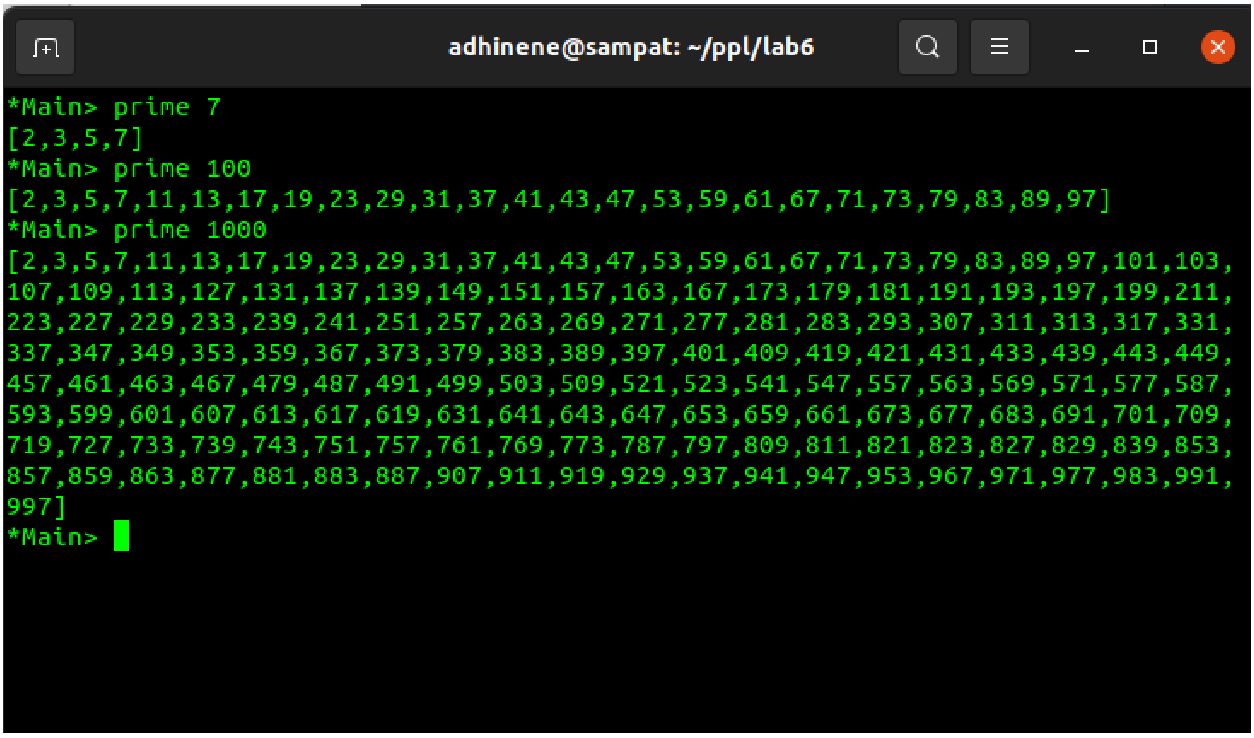
```
*Main> factor 625
[1,5,25,125,625]
*Main> factor 9761
[1,43,227,9761]
*Main> factor 20
[1,2,4,5,10,20]
*Main>
```

Q2)

2. Using list comprehension, define a function that **returns the list of all prime numbers** up to a given limit **n**.

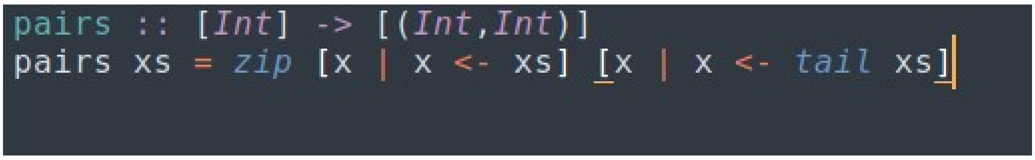


```haskell
prime :: Int -> [Int]
prime x = [k | k <- [2..x],p k <=2]
          where p d = length [f | f <- [1..d],d `mod` f==0]
```

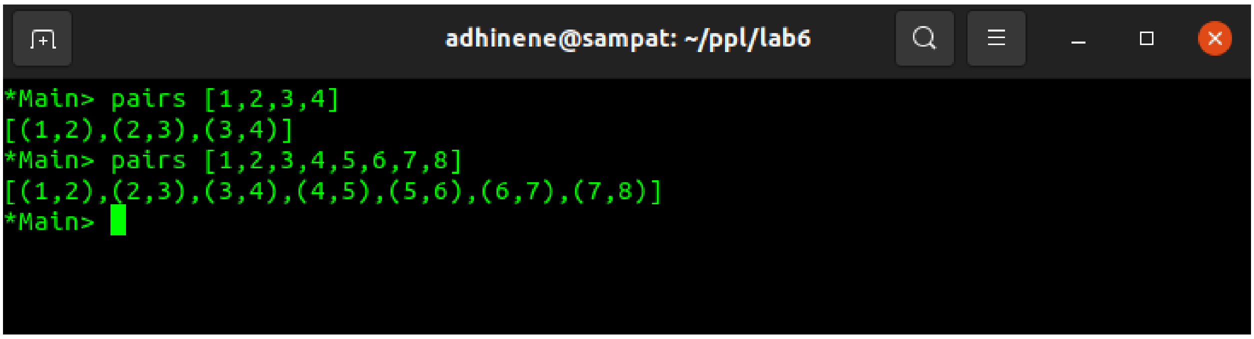


```
*Main> prime 7
[2,3,5,7]
*Main> prime 100
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]
*Main> prime 1000
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,
107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,
223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331,
337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,
457,461,463,467,479,487,491,499,503,509,521,523,541,547,557,563,569,571,577,587,
593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,
719,727,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,
857,859,863,877,881,883,887,907,911,919,929,937,941,947,953,967,971,977,983,991,
997]
*Main> 
```

3. Define a function **pairs** that uses the **zip** function for **returning the list of all pairs of adjacent elements from a list**.
   Example – **Input**: pairs [1,2,3,4] **Output**: [(1,2),(2,3),(3,4)].

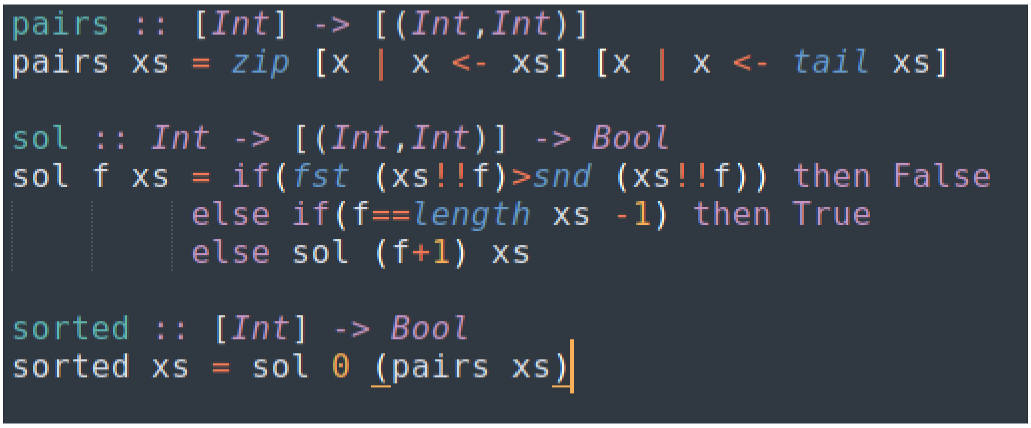


```haskell
pairs :: [Int] -> [(Int,Int)]
pairs xs = zip [x | x <- xs] [x | x <- tail xs]
```

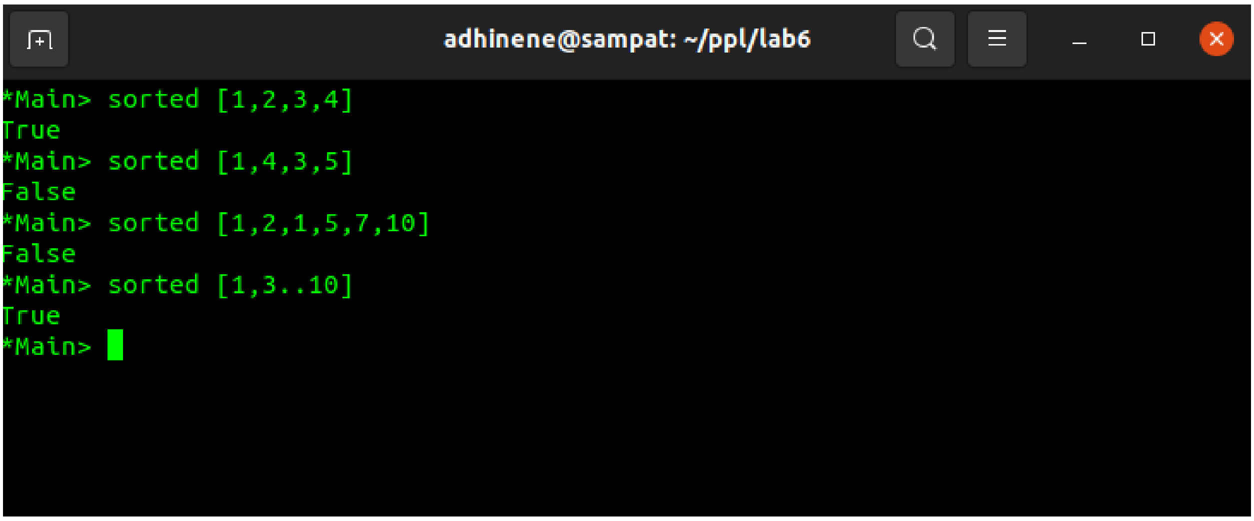


```
*Main> pairs [1,2,3,4]
[(1,2),(2,3),(3,4)]
*Main> pairs [1,2,3,4,5,6,7,8]
[(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8)]
*Main> 
```

4. Using the **pairs** function in **Q3** define a function **sorted** that decides if the **elements in a list are sorted** [here we are checking list which is formed by pairs function].
   Example – **Input**: sorted [1,2,3,4] **Output**: True

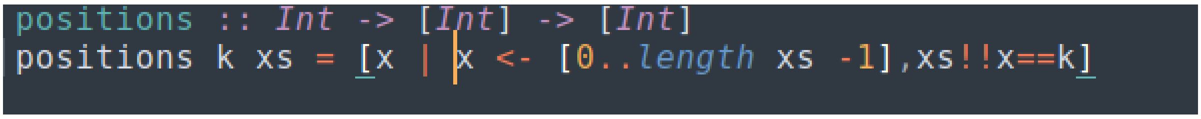


```haskell
pairs :: [Int] -> [(Int,Int)]
pairs xs = zip [x | x <- xs] [x | x <- tail xs]

sol :: Int -> [(Int,Int)] -> Bool
sol f xs = if(fst (xs!!f)>snd (xs!!f)) then False
         else if(f==length xs -1) then True
         else sol (f+1) xs

sorted :: [Int] -> Bool
sorted xs = sol 0 (pairs xs)
```

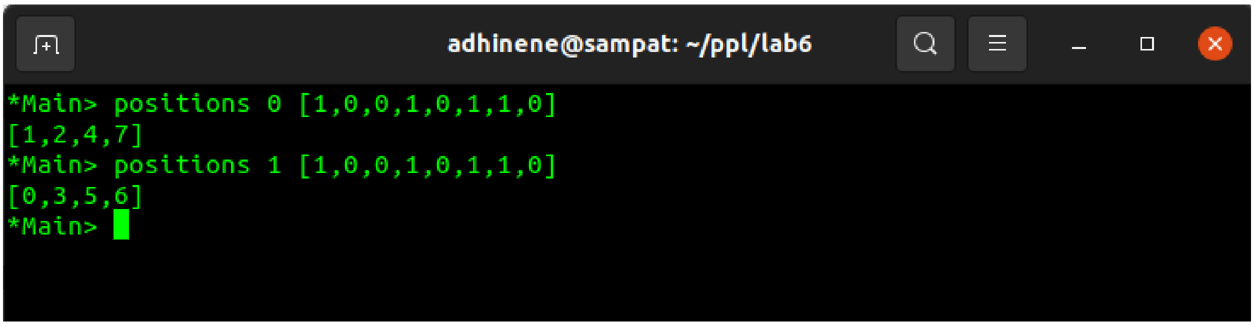


```
*Main> sorted [1,2,3,4]
True
*Main> sorted [1,4,3,5]
False
*Main> sorted [1,2,1,5,7,10]
False
*Main> sorted [1,3..10]
True
*Main> 
```

5. Using list comprehension, define a function **positions** using **zip** function which **will return the list of all positions of a value in a list**.
   Example – **Input**: positions 0 [1,0,0,1,0,1,1,0] **Output**: [1,2,4,7].

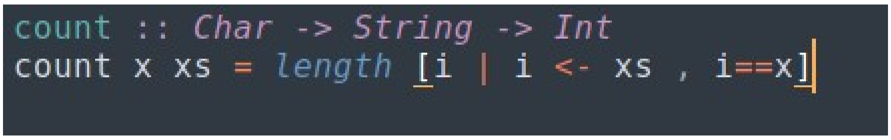


```haskell
positions :: Int -> [Int] -> [Int]
positions k xs = [x | x <- [0..length xs -1],xs!!x==k]
```

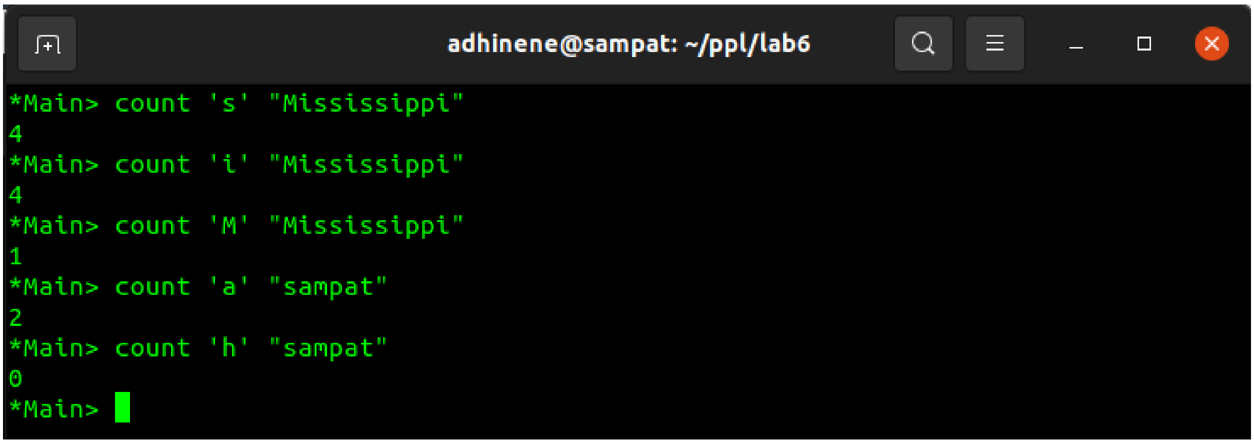


```
*Main> positions 0 [1,0,0,1,0,1,1,0]
[1,2,4,7]
*Main> positions 1 [1,0,0,1,0,1,1,0]
[0,3,5,6]
*Main> 
```

6. Using list comprehension, define a function **count** to get the **number of times a character occurs in a String**.
   Example – **Input**: count 's' "Mississippi" **Output**: 4.

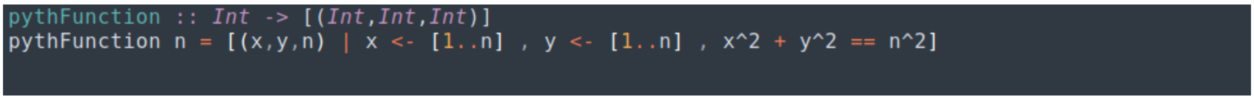


```haskell
count :: Char -> String -> Int
count x xs = length [i | i <- xs , i==x]
```

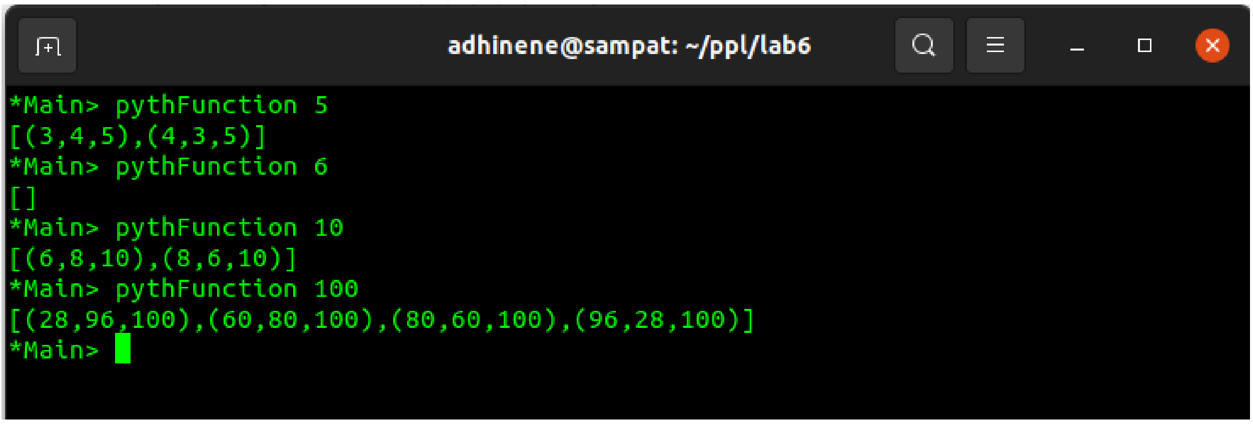


```
*Main> count 's' "Mississippi"
4
*Main> count 'i' "Mississippi"
4
*Main> count 'M' "Mississippi"
1
*Main> count 'a' "sampat"
2
*Main> count 'h' "sampat"
0
*Main> 
```

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].
   Example – **Input**: pythFunction 5 **Output**: [(3,4,5),(4,3,5)].

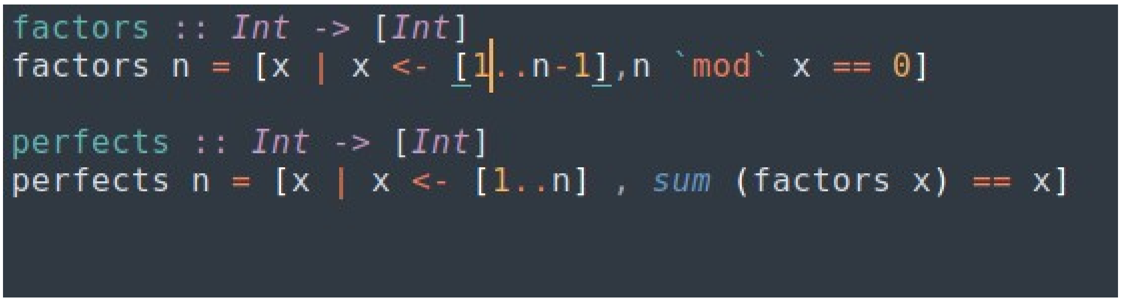


```
pythFunction :: Int -> [(Int,Int,Int)]
pythFunction n = [(x,y,n) | x <- [1..n] , y <- [1..n] , x^2 + y^2 == n^2]
```

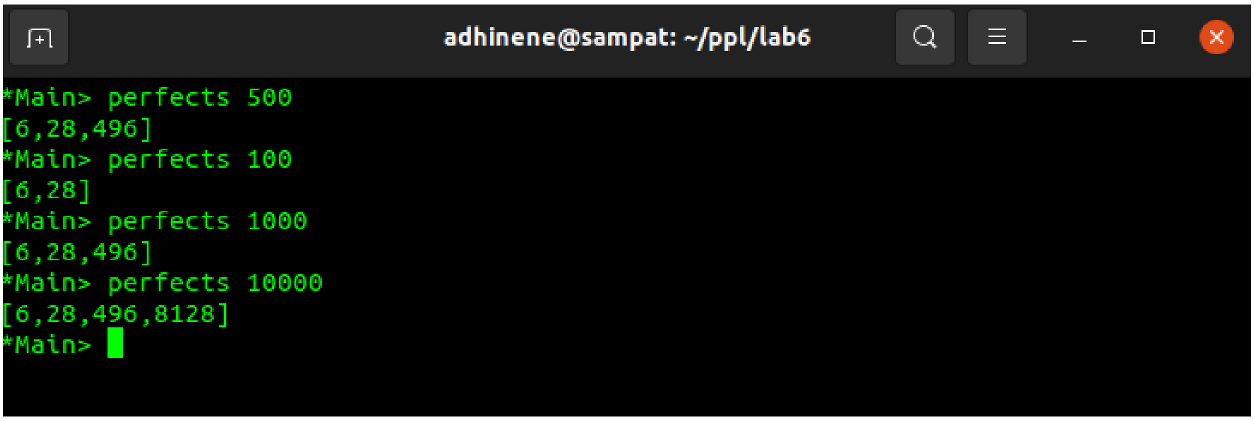


```
*Main> pythFunction 5
[(3,4,5),(4,3,5)]
*Main> pythFunction 6
[]
*Main> pythFunction 10
[(6,8,10),(8,6,10)]
*Main> pythFunction 100
[(28,96,100),(60,80,100),(80,60,100),(96,28,100)]
*Main> 
```

8. A perfect number is a positive integer which is equal to the sum of all its factors, excluding the number itself. Using list comprehension, define a function **perfects :: Int->Int** that **returns all the perfect numbers up to a given limit n**.
   Example – **Input**: perfects 500 **Output**: [6,28,496].



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

perfects :: Int -> [Int]
perfects n = [x | x <- [1..n] , sum (factors x) == x]
```

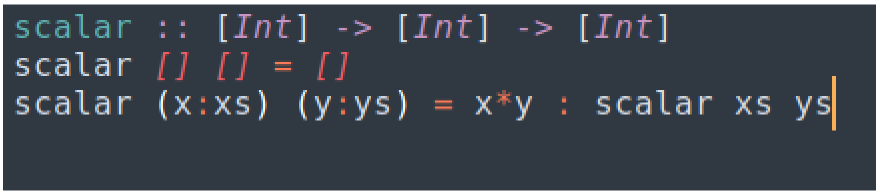


```
*Main> perfects 500
[6,28,496]
*Main> perfects 100
[6,28]
*Main> perfects 1000
[6,28,496]
*Main> perfects 10000
[6,28,496,8128]
*Main> 
```

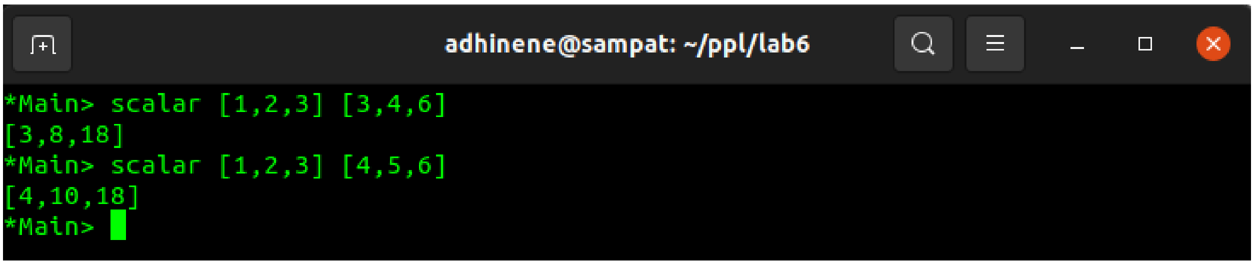
9. Using list comprehension define a function **scalar** to find the **scalar product of list elements of two lists xs and ys of length n**.

$$\sum_{i=0}^{n-1} (xs_i * ys_i)$$

Example – **Input**: scalar [1,2,3] [3,4,6] **Output**: [3,8,18].

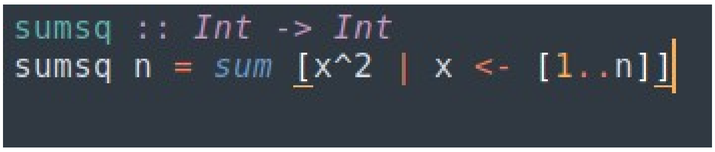


```
scalar :: [Int] -> [Int] -> [Int]
scalar [] [] = []
scalar (x:xs) (y:ys) = x*y : scalar xs ys
```

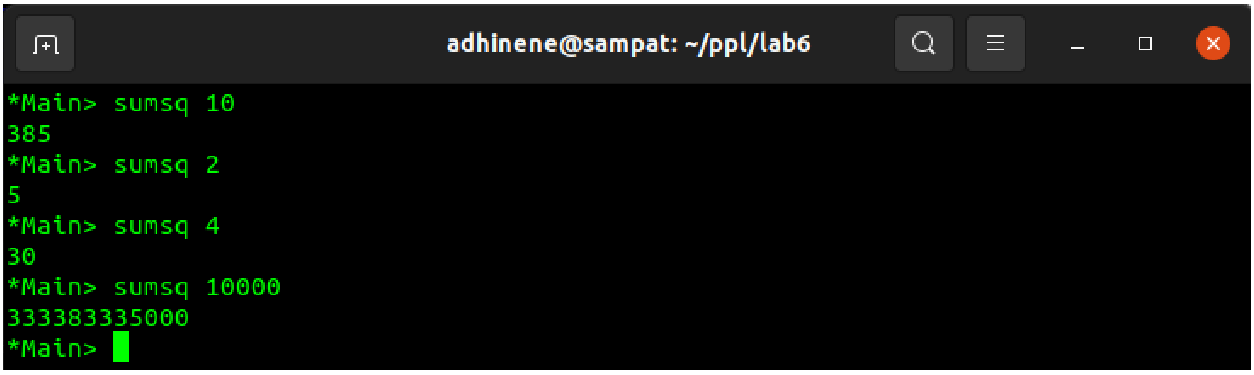


```
*Main> scalar [1,2,3] [3,4,6]
[3,8,18]
*Main> scalar [1,2,3] [4,5,6]
[4,10,18]
*Main> 
```

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**.

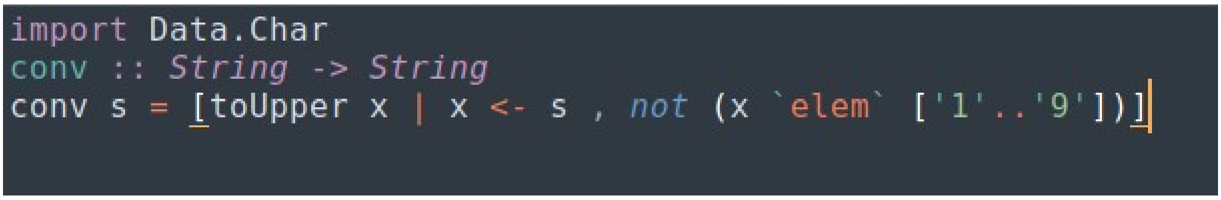


```
sumsq :: Int -> Int
sumsq n = sum [x^2 | x <- [1..n]]
```

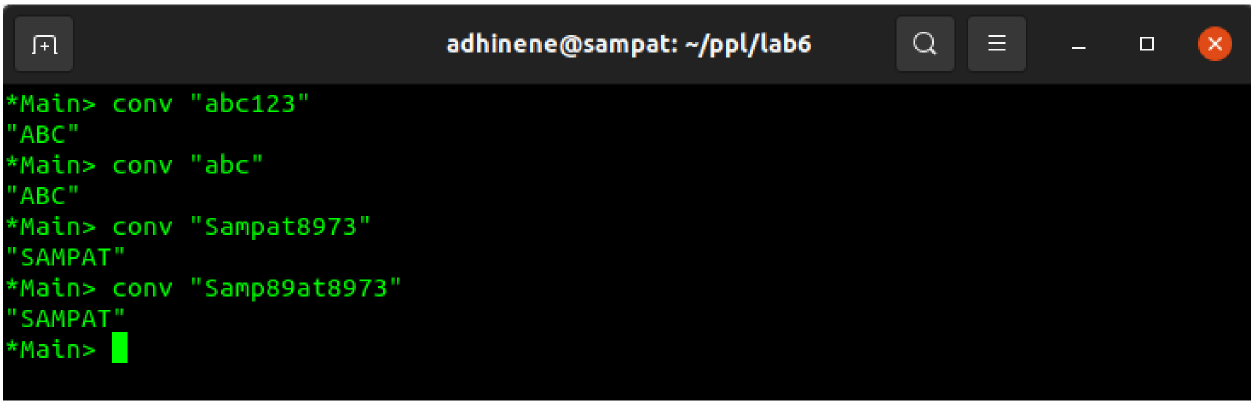


```
*Main> sumsq 10
385
*Main> sumsq 2
5
*Main> sumsq 4
30
*Main> sumsq 10000
333383335000
*Main> 
```

11. Using string comprehension, **convert every character in string to uppercase and remove any digits in it**.

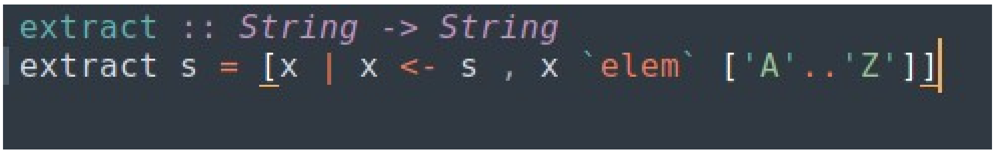


```haskell
import Data.Char
conv :: String -> String
conv s = [toUpper x | x <- s , not (x `elem` ['1'..'9'])]
```

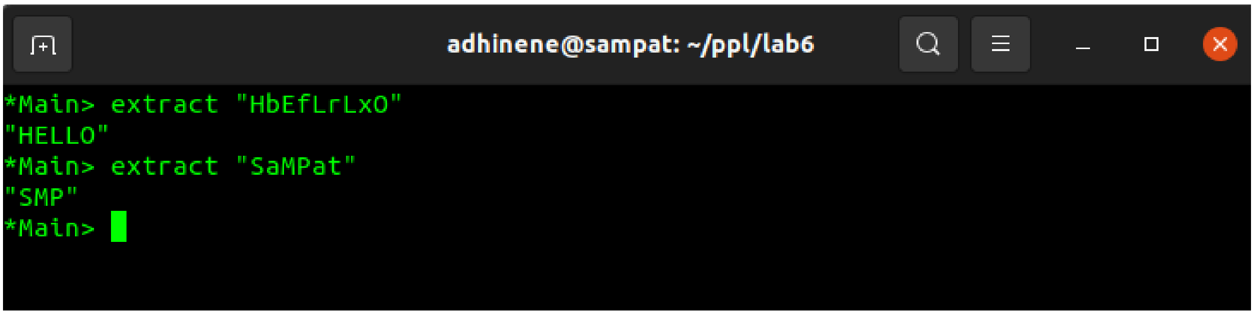


```
*Main> conv "abc123"
"ABC"
*Main> conv "abc"
"ABC"
*Main> conv "Sampat8973"
"SAMPAT"
*Main> conv "Samp89at8973"
"SAMPAT"
*Main> 
```

12. Define a function that **extracts the upper case letters only**. Given the input “HbEfLrLxO”, your function will return “HELLO”.

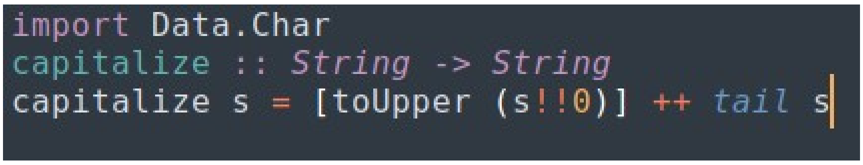


```haskell
extract :: String -> String
extract s = [x | x <- s , x `elem` ['A'..'Z']]
```

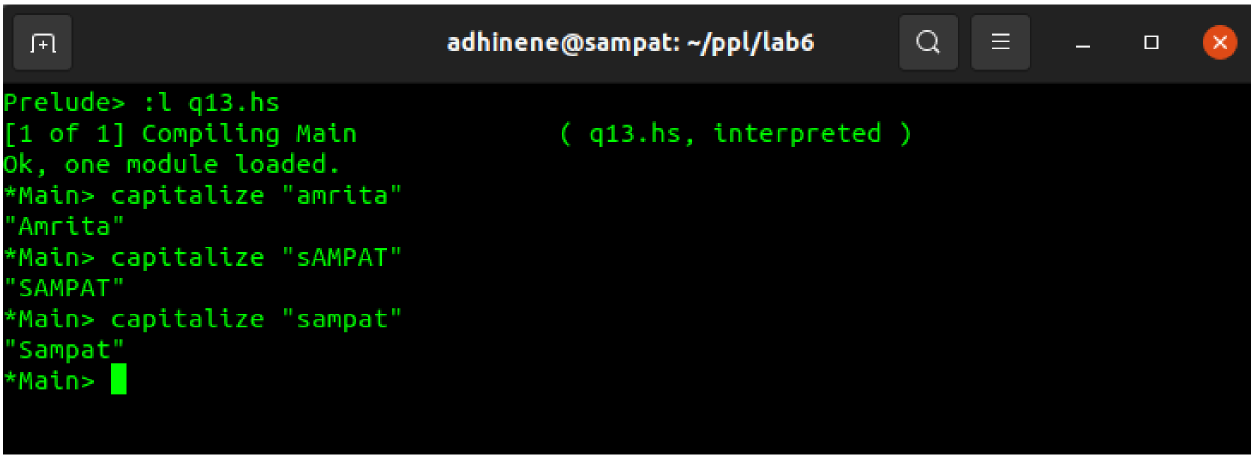


```
*Main> extract "HbEfLrLxO"
"HELLO"
*Main> extract "SaMPat"
"SMP"
*Main> 
```

13. Define a function that will **capitalize the first letter of a String and return the entire String**. For example, if given the argument “amrita,” it will return “Amrita.”

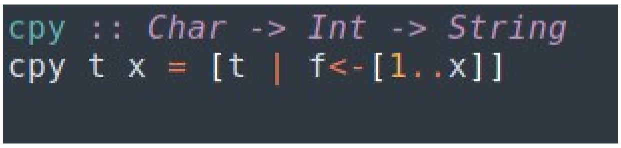


```haskell
import Data.Char
capitalize :: String -> String
capitalize s = [toUpper (s!!0)] ++ tail s
```

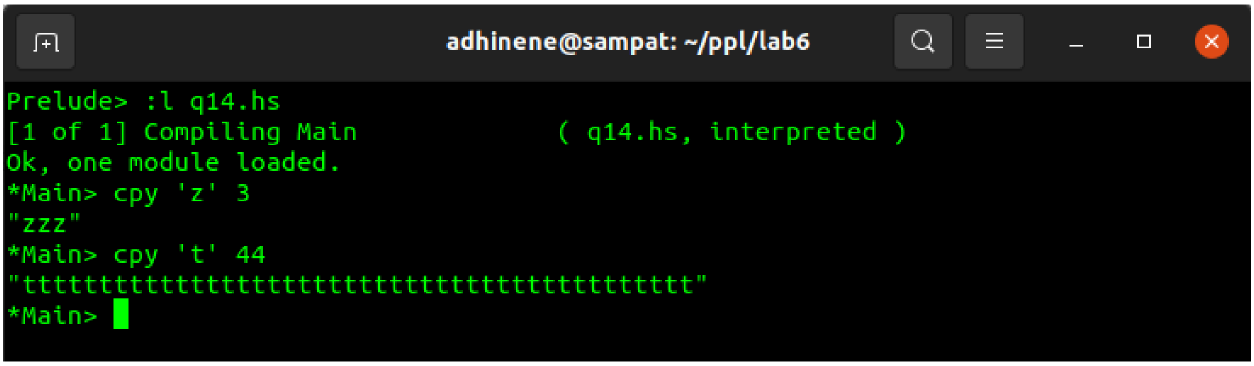


```
Prelude> :l q13.hs
[1 of 1] Compiling Main             ( q13.hs, interpreted )
Ok, one module loaded.
*Main> capitalize "amrita"
"Amrita"
*Main> capitalize "sAMPAT"
"SAMPAT"
*Main> capitalize "sampat"
"Sampat"
*Main> 
```

14. Define a function **cpy** to **make a string of n characters**. ***Example***: **Input** cpy ‘z’ 3 **Output** “zzz”.

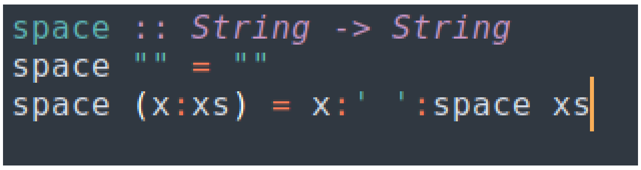


```haskell
cpy :: Char -> Int -> String
cpy t x = [t | f<-[1..x]]
```

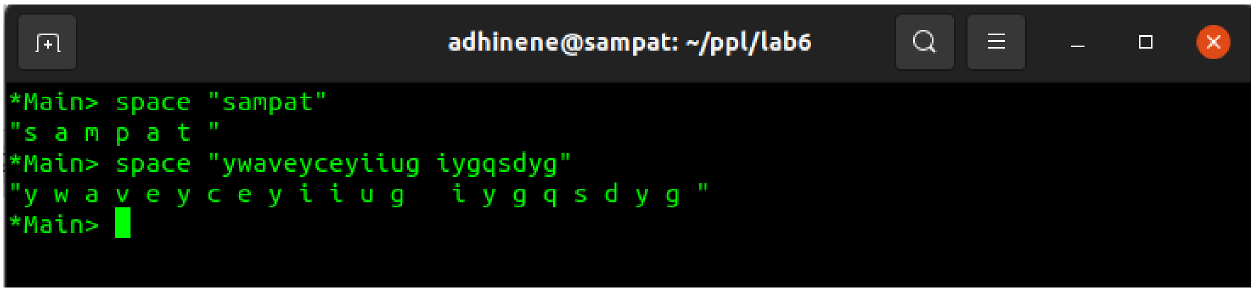


```
Prelude> :l q14.hs
[1 of 1] Compiling Main             ( q14.hs, interpreted )
Ok, one module loaded.
*Main> cpy 'z' 3
"zzz"
*Main> cpy 't' 44
"tttttttttttttttttttttttttttttttttttttttttttt"
*Main> 
```

15. Define a function to **place space characters between characters in a string**.



```haskell
space :: String -> String
space "" = ""
space (x:xs) = x:' ':space xs
```



```
*Main> space "sampat"
"s a m p a t "
*Main> space "ywaveyceyiiug iygqsdyg"
"y w a v e y c e y i i u g   i y g q s d y g "
*Main> 
```