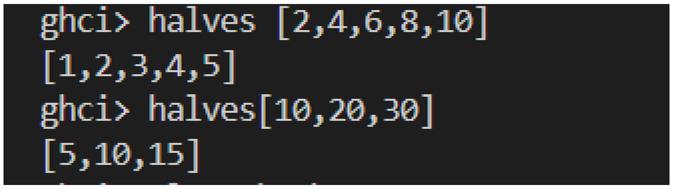
ASSIGNMENT 3
LAB TASK-2

NAME: B.THARUN
ROLL NO: 422116
SECTION: A

1.Implement a function halves that takes a list of integers and divides each element of the listin two

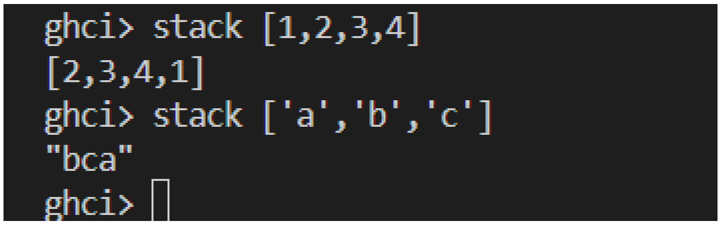
```
halves :: [Int] -> [Float]
halves xs = [fromIntegral x / 2 | x <- xs]
```



```
ghci> halves [2,4,6,8,10]
[1,2,3,4,5]
ghci> halves[10,20,30]
[5,10,15]
```

2.Implement a function stack that takes the first element of a list and moves it to the back

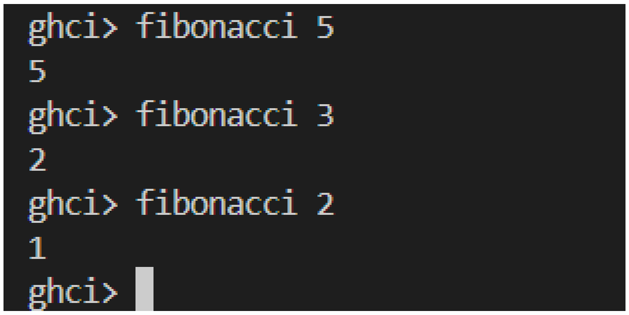
```
stack :: [a] -> [a]
stack [] = []
stack (x:xs) = xs ++ [x]
```



```
ghci> stack [1,2,3,4]
[2,3,4,1]
ghci> stack ['a','b','c']
"bca"
ghci>
```

3.Implement a function that computes the nth Fibonacci no

```
fibonacci :: Int -> Int
fibonacci 0 = 0
fibonacci 1 = 1
fibonacci n = fibonacci (n - 1) + fibonacci (n - 2)
```

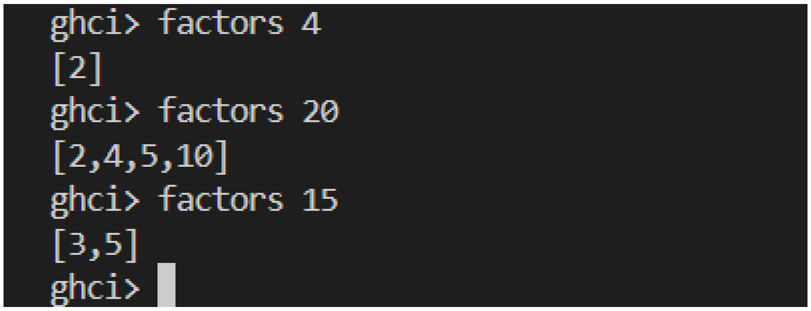


```
ghci> fibonacci 5
5
ghci> fibonacci 3
2
ghci> fibonacci 2
1
ghci>
```

4.implement a function factors that takes an Int and returns a list off all its factors (i.e. all the

Int’s bigger than 1 and less than that Int that are divisible without a remainder)

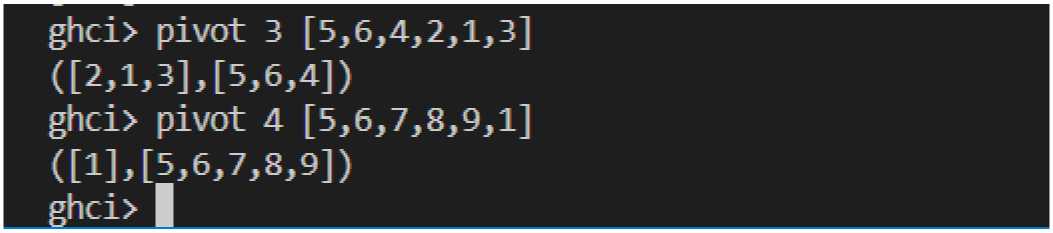
```
factors :: Int -> [Int]
factors n = [x | x <- [2..n-1], n `mod` x == 0]
```



```
ghci> factors 4
[2]
ghci> factors 20
[2,4,5,10]
ghci> factors 15
[3,5]
ghci>
```

5.Implement a function pivot that takes a value and a list, then returns two lists in a tuple, with the first list being all elements <= to the value, and the second list being all elements >the value I.e. pivot 3 [5,6,4,2,1,3]= ([2,1,3],[5,6,4])

```
pivot :: Ord a => a -> [a] -> ([a], [a])
pivot val xs = (filter (<= val) xs, filter (> val) xs)
```



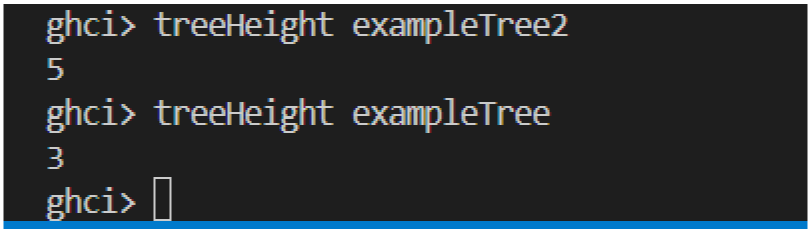
```
ghci> pivot 3 [5,6,4,2,1,3]
([2,1,3],[5,6,4])
ghci> pivot 4 [5,6,7,8,9,1]
([1],[5,6,7,8,9])
ghci>
```

6.Implement the function treeHeight that returns the largest height of a Tree

7. -- E.x.

a
**/ **
b c
**/ **
d e
has a height of 3 (elements d and e are both at &quot;height&quot; 3 in the tree)
NOTE theEmpty Tree is of height 0

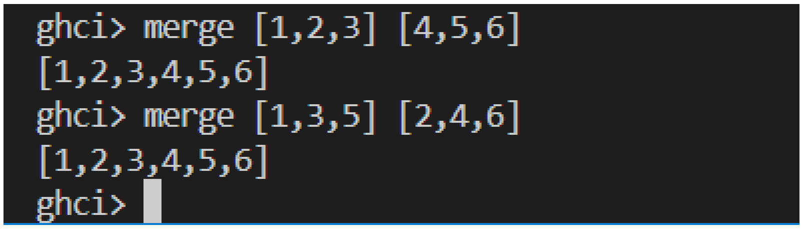
```
data Tree a = Empty | Node a (Tree a) (Tree a)
    deriving (Show)

treeHeight :: Tree a -> Int
treeHeight Empty = 0
treeHeight (Node _ left right) = 1 + max (treeHeight left) (treeHeight right)

-- Example tree:
--     a
--    / \
--  b   c
--  / \
-- d  e
exampleTree = Node 'a' (Node 'b' (Node 'd' Empty Empty) (Node 'e' Empty Empty)) (Node 'c' Empty Empty)
--       a
--     / \
--    b  c
--  / \
--  d  e
-- /
--f
--/
--g

exampleTree2 :: Tree Char
exampleTree2 = Node 'a'
            (Node 'b'
               (Node 'd'
                  (Node 'f'
                     (Node 'g' Empty Empty)
                     Empty
                  )
                  Empty
               )
               (Node 'e' Empty Empty)
            )
            (Node 'c' Empty Empty)
```



```
ghci> treeHeight exampleTree2
5
ghci> treeHeight exampleTree
3
ghci> 
```

8.Implement the function merge that takes two lists that (assuming both lists are alreadysorted) merges them together into a sorted list

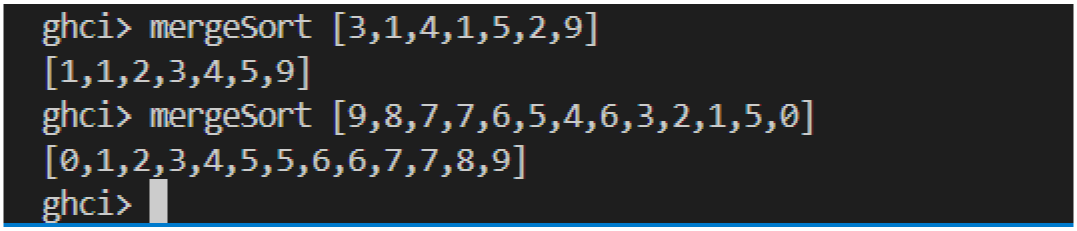
```
merge :: Ord a => [a] -> [a] -> [a]
merge [] ys = ys
merge xs [] = xs
merge (x:xs) (y:ys)
   | x <= y    = x : merge xs (y:ys)
   | otherwise = y : merge (x:xs) ys
```



```
ghci> merge [1,2,3] [4,5,6]
[1,2,3,4,5,6]
ghci> merge [1,3,5] [2,4,6]
[1,2,3,4,5,6]
ghci>
```

9.Implement the function mergeSort that sorts a list by reclusively splitting a list, and mergingthe sorted lists back together. NOTE singleton and empty lists are already sorted

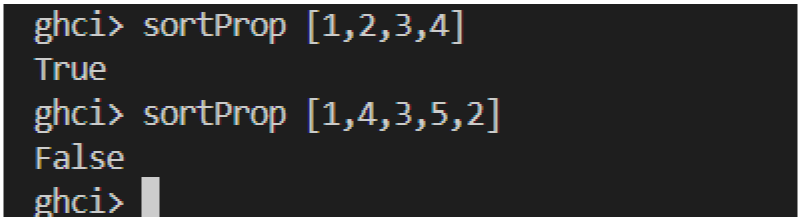
```
mergeSort :: Ord a => [a] -> [a]
mergeSort [] = []
mergeSort [x] = [x]
mergeSort xs = merge (mergeSort left) (mergeSort right)
  where
    (left, right) = splitAt (length xs `div` 2) xs
```



```
ghci> mergeSort [3,1,4,1,5,2,9]
[1,1,2,3,4,5,9]
ghci> mergeSort [9,8,7,7,6,5,4,6,3,2,1,5,0]
[0,1,2,3,4,5,5,6,6,7,7,8,9]
ghci>
```

10.Implement the function sortProp that tests if a list is sorted or not. NOTE you can use this with QuickCheck to test your mergeSort function by calling quickCheck (sortProp . mergeSort)

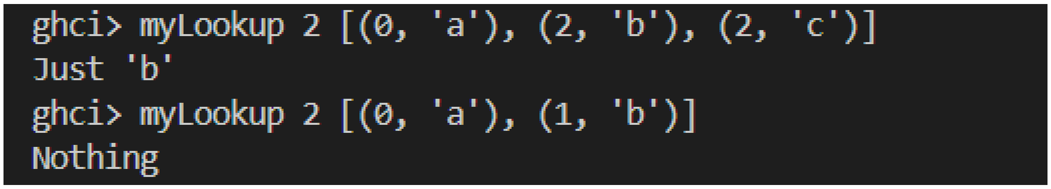
```
sortProp :: Ord a => [a] -> Bool
sortProp [] = True
sortProp [_] = True
sortProp (x:y:ys) = x <= y && sortProp (y:ys)
```



```
ghci> sortProp [1,2,3,4]
True
ghci> sortProp [1,4,3,5,2]
False
ghci>
```

11.Implement the function lookup that takes a list of tuples, where the first element of the tuple serves as a key and the second element a value (a list like this is also known as a dictionary), and a key value, then looks up the first occurring element corresponding to thatkey. The return value is wrapped in the Maybe type, so if the key doesn't occur anywhere inthe list the function returns Nothing
E.x. lookup 2 [(0,'a'),(1,'b')] ==
Nothing lookup 2 [(0,'a'),(2,'b'),(2,'c')]
== Just 'b';

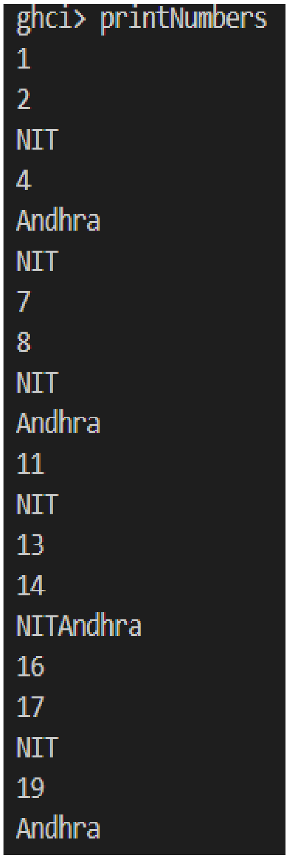
```
myLookup :: Eq a => a -> [(a, b)] -> Maybe b
myLookup _ [] = Nothing
myLookup key ((k, v):xs)
    | key == k  = Just v
    | otherwise = myLookup key xs
```



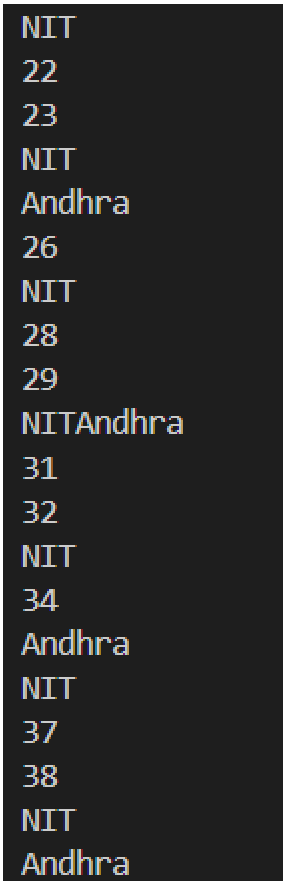
```
ghci> myLookup 2 [(0, 'a'), (2, 'b'), (2, 'c')]
Just 'b'
ghci> myLookup 2 [(0, 'a'), (1, 'b')]
Nothing
```

12.Write a program that prints the integers from 1 to 100 (inclusive). But: for multiples ofthree, print NIT (instead of the number) for multiples of five, print Andhra (instead of
the number) for multiples of both three and five, print NITAndhra (instead of thenumber)

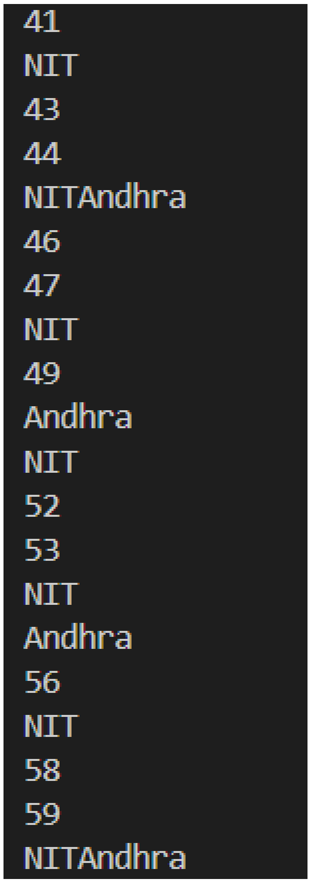
```
printNumbers :: IO ()
printNumbers = mapM_ putStrLn [ result x | x <- [1..100] ]
  where
    result x
      | x `mod` 15 == 0 = "NITAndhra"
      | x `mod` 3 == 0  = "NIT"
      | x `mod` 5 == 0  = "Andhra"
      | otherwise       = show x
```



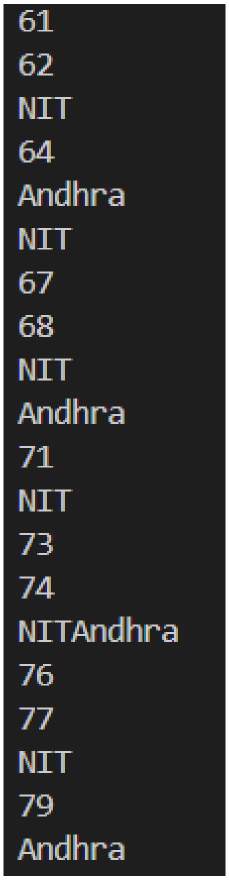
```
ghci> printNumbers
1
2
NIT
4
Andhra
NIT
7
8
NIT
Andhra
11
NIT
13
14
NITAndhra
16
17
NIT
19
Andhra
```



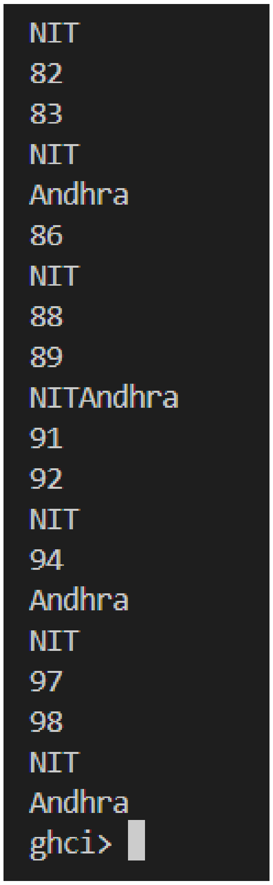
```
NIT
22
23
NIT
Andhra
26
NIT
28
29
NITAndhra
31
32
NIT
34
Andhra
NIT
37
38
NIT
Andhra
```



```
41
NIT
43
44
NITAndhra
46
47
NIT
49
Andhra
NIT
52
53
NIT
Andhra
56
NIT
58
59
NITAndhra
```



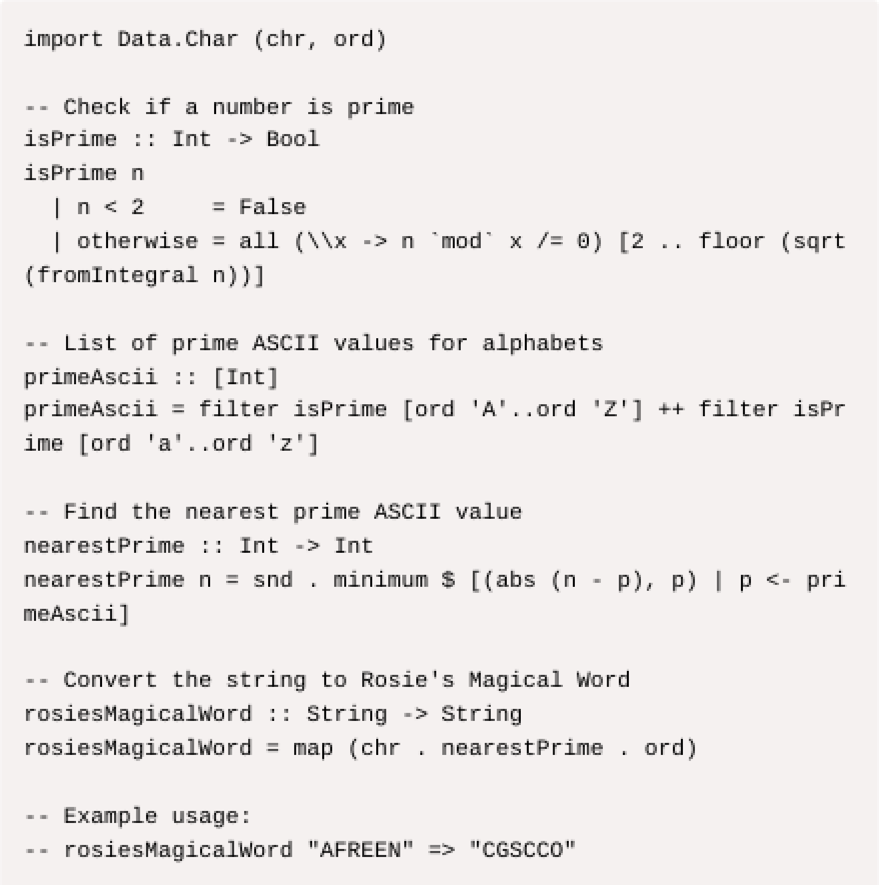
```
61
62
NIT
64
Andhra
NIT
67
68
NIT
Andhra
71
NIT
73
74
NITAndhra
76
77
NIT
79
Andhra
```



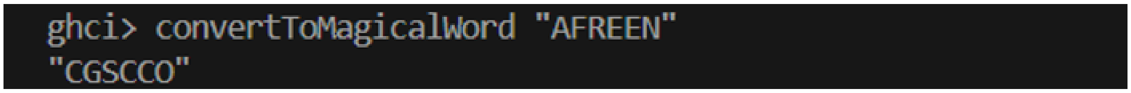
```
NIT
82
83
NIT
Andhra
86
NIT
88
89
NITAndhra
91
92
NIT
94
Andhra
NIT
97
98
NIT
Andhra
ghci>
```

13.Rosie has recently learned about ASCII values. She is very fond of experimenting. With hisknowledge of ASCII values and characters. She has developed a special word and named it Rosie’s Magical word. A word that consists of alphabets whose ASCII value is a prime
number is Rosie’s Magical word. An alphabet is Rosie’s Magical alphabet if its ASCII value isprime. convert The given strings to Rosie’s Magical Word.
Rules for converting:
1.Each character should be replaced by The nearest Rosie’s Magical alphabet.
2.If the character is equidistant with 2 Magical alphabets. The one with a lower ASCII valuewill be considered as its replacement.
Input:
AFREEN
Output:
CGSCCO

Explanation
ASCII values of alphabets in AFREEN are 65, 70, 82, 69, 69 and 78 respectively which are converted to CGSCCO with ASCII values 67, 71, 83, 67, 67, 79 respectively. All such ASCIIvalues are prime numbers.



```haskell
import Data.Char (chr, ord)

-- Check if a number is prime
isPrime :: Int -> Bool
isPrime n
  | n < 2     = False
  | otherwise = all (\\x -> n `mod` x /= 0) [2 .. floor (sqrt (fromIntegral n))]

-- List of prime ASCII values for alphabets
primeAscii :: [Int]
primeAscii = filter isPrime [ord 'A'..ord 'Z'] ++ filter isPrime [ord 'a'..ord 'z']

-- Find the nearest prime ASCII value
nearestPrime :: Int -> Int
nearestPrime n = snd . minimum $ [(abs (n - p), p) | p <- primeAscii]

-- Convert the string to Rosie's Magical Word
rosiesMagicalWord :: String -> String
rosiesMagicalWord = map (chr . nearestPrime . ord)

-- Example usage:
-- rosiesMagicalWord "AFREEN" => "CGSCCO"
```



```
ghci> convertToMagicalWord "AFREEN"
"CGSCCO"
```

14.n people standing in a circle in order from 1 to n. if n=5 and then No. 1 has a sword. He killsthe next person (i.e. No. 2) and gives the sword to the next (i.e. No. 3). All people do the same until only 1 survives. Which number survives at the last? Note: Initially knife will bewith the first person (i.e. No. 1)

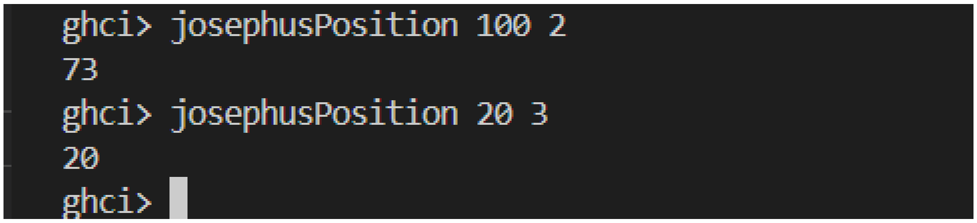
Input:

100

Output:

73

```
josephus :: Int -> Int -> Int
josephus 1 _ = 0
josephus n k = (josephus (n - 1) k + k) `mod` n

josephusPosition :: Int -> Int -> Int
josephusPosition n k = josephus n k + 1
```

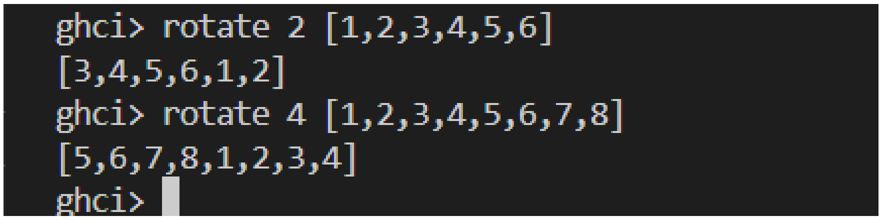


```
ghci> josephusPosition 100 2
73
ghci> josephusPosition 20 3
20
ghci>
```

15.Write a function to rotate a list in Haskell by giving a K value.Input: [1, 2, 3, 4, 5, 6, 7] K=2

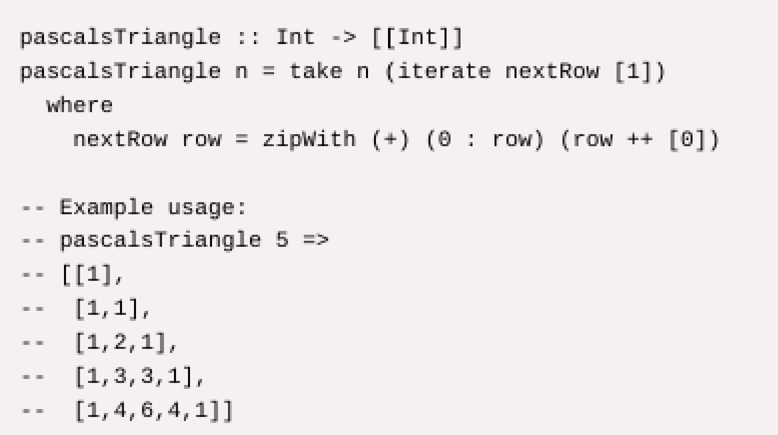
Output:[3, 4, 5, 6, 7, 1, 2]

```
rotate :: Int -> [a] -> [a]
rotate k xs = drop k xs ++ take k xs
```

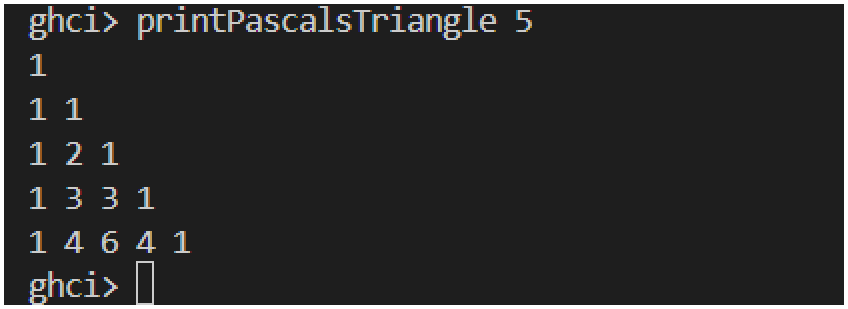


```
ghci> rotate 2 [1,2,3,4,5,6]
[3,4,5,6,1,2]
ghci> rotate 4 [1,2,3,4,5,6,7,8]
[5,6,7,8,1,2,3,4]
ghci>
```

16.Compute Pascal&#39;s triangle up to a given number of rows.In Pascal&#39;s Triangleeach number is
computed by adding the numbers to the right and left of the current position in the previousrow.
Input:5
Output
:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

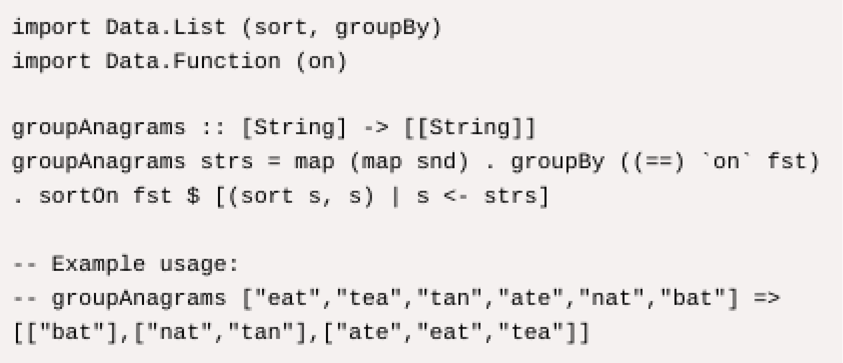


```haskell
pascalsTriangle :: Int -> [[Int]]
pascalsTriangle n = take n (iterate nextRow [1])
  where
    nextRow row = zipWith (+) (0 : row) (row ++ [0])

-- Example usage:
-- pascalsTriangle 5 =>
-- [[1],
--  [1,1],
--  [1,2,1],
--  [1,3,3,1],
--  [1,4,6,4,1]]
```

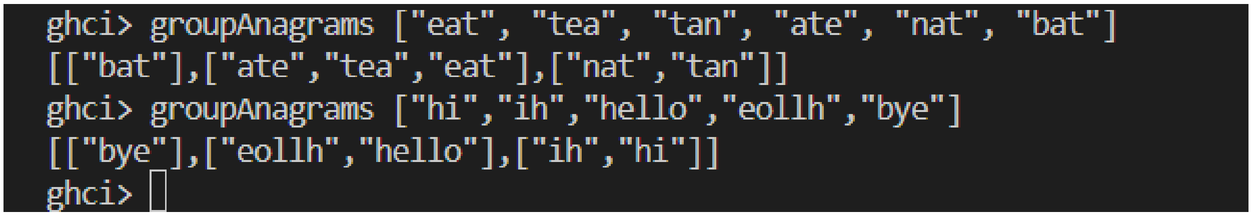


```
ghci> printPascalsTriangle 5
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
ghci> 
```

17. Given an array of strings strs, group the anagrams together. You can return the answer inany order. An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.
Input:
[&quot;eat&quot;,&quot;tea&quot;,&quot;tan&quot;,&quot;ate&quot;,&quot;nat&quot;,&quot;bat& quot;]
Output:
[[&quot;bat&quot;],[&quot;nat&quot;,&quot;tan&quot;],[&quot;ate&quot;,&quot;eat&quot;,&quot;t ea&quot;]]

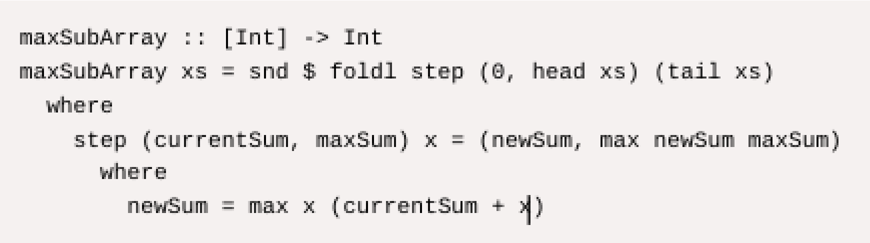


```
import Data.List (sort, groupBy)
import Data.Function (on)

groupAnagrams :: [String] -> [[String]]
groupAnagrams strs = map (map snd) . groupBy ((==) `on` fst)
. sortOn fst $ [(sort s, s) | s <- strs]

-- Example usage:
-- groupAnagrams ["eat","tea","tan","ate","nat","bat"] =>
[["bat"],["nat","tan"],["ate","eat","tea"]]
```

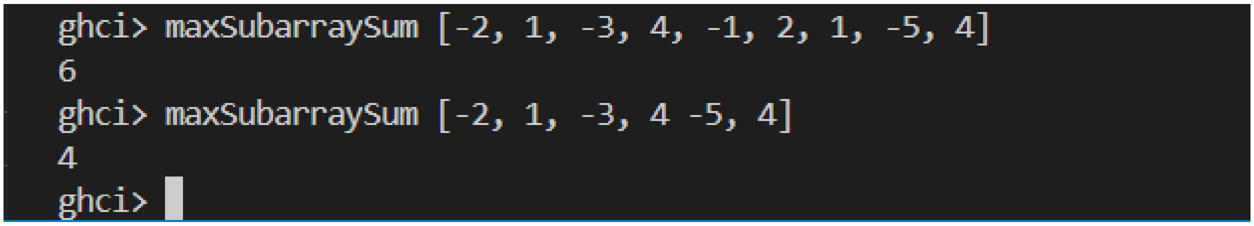


```
ghci> groupAnagrams ["eat", "tea", "tan", "ate", "nat", "bat"]
[["bat"],["ate","tea","eat"],["nat","tan"]]
ghci> groupAnagrams ["hi","ih","hello","eollh","bye"]
[["bye"],["eollh","hello"],["ih","hi"]]
ghci> 
```

18.Given an integer array nums, find the contiguous subarray (containing at least one number)which has the largest sum and return its sum.
A subarray is a contiguous part of an array.Input:[-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.



```
maxSubArray :: [Int] -> Int
maxSubArray xs = snd $ foldl step (0, head xs) (tail xs)
  where
    step (currentSum, maxSum) x = (newSum, max newSum maxSum)
      where
        newSum = max x (currentSum + x)
```



```
ghci> maxSubarraySum [-2, 1, -3, 4, -1, 2, 1, -5, 4]
6
ghci> maxSubarraySum [-2, 1, -3, 4 -5, 4]
4
ghci>
```