Code (Tuesday Week 1)

Haskell

```haskell
import Data.Char (toLower)
import Data.List (sortBy, group, sort)

toCartesian
  :: (Double, Double) -> (Double,Double)
toCartesian (r, theta) = (x, y)
  where
    y = r * sin theta
    x = r * cos theta

breakIntoWords :: String -> [String]
breakIntoWords = words

convertToLowercase :: [String] -> [String]
convertToLowercase = map (map toLower)

sortWords :: [String] -> [String]
sortWords = sort

countAdjacentRuns :: [String] -> [Run]
countAdjacentRuns = toRuns
                  . groupAdjacentRuns

groupAdjacentRuns :: [String] -> [[String]]
groupAdjacentRuns = group

toRuns :: [[String]] -> [Run]
toRuns = map (\ls -> (head ls, length ls))

type Run = (String, Int)

sortByRun :: [Run] -> [Run]
sortByRun = sortBy (\(w1,l1) (w2, l2)
                     -> compare l2 l1)

takeFirst :: Int -> [Run] -> [Run]
takeFirst n = take n

generateReport :: [Run] -> String
generateReport
  = unlines
  . map (\(w, l) -> w ++ ":" ++ show l)

commonWords :: Int -> String -> String
```

```haskell
commonWords n
  = generateReport
  . takeFirst n
  . sortByRun
  . countAdjacentRuns
  . sortWords
  . convertToLowercase
  . breakIntoWords

-- Renamed from (++) to avoid clashing with stdlib
(+++) :: [a] -> [a] -> [a]
[] +++ ys = ys
(x:xs) +++ ys = x : (xs +++ ys)

sum' :: [Int] -> Int
sum' [] = 0
sum' (x:xs) = x + sum' xs

concat' :: [[a]] -> [a]
concat' [] = []
concat' (x:xs) = x ++ concat xs

filter' :: (a -> Bool) -> [a] -> [a]
filter' p [] = []
filter' p (x:xs)
  | p x       = x : filter' p xs
  | otherwise = filter' p xs
```