```
c :: [Int] -> [Int]
c (x:xs)  =  [ x | (x,y) <- zip (x:xs) xs, x == y ]

d :: [Int] -> [Int]
d [x]       =  []
d (x:y:zs) | x==y       = x : d (y:zs)
           | otherwise  = d (y:zs)

isInRange :: Int -> Bool
isInRange x = 0 <= x && x <= 100

isEven:: Int -> Bool
isEven n = n `mod` 2 == 0

r :: [Int] -> Bool
--r xs = foldr (&&) True (map isEven (filter isInRange xs))
r xs = foldr (&&) True (map (\y -> mod y 2 == 0) (filter (\x -> 0 <= x
&& x <= 100) xs))

---
type Fabricante = String
type Potencia = Float
data Lampada = Compacta Fabricante Potencia
               | Incand Fabricante Potencia

instance Show Lampada where
   show (Compacta f p) = "Compacta" ++ " " ++  f ++ " " ++ show p
   show  (Incand i h) = "Incand " ++ " " ++  i ++ " " ++ show h

instance Eq Lampada where
    (Compacta f1 p1) == (Compacta f2 p2) = f1 == f2 && p1 == p2
    (Incand f1 p1) == (Incand f2 p2) = f1 == f2 && p1 == p2
    _ == _ = False
```