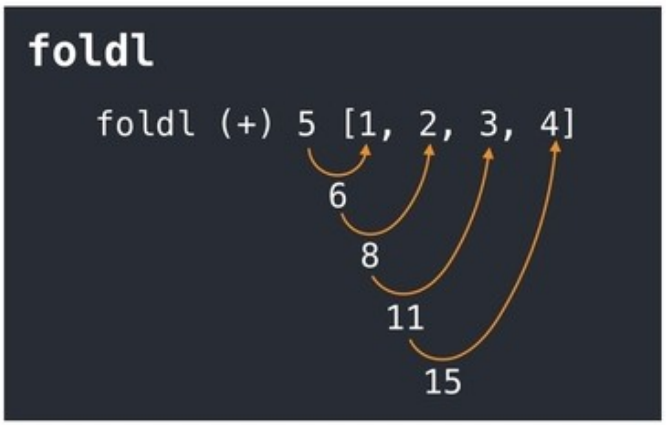
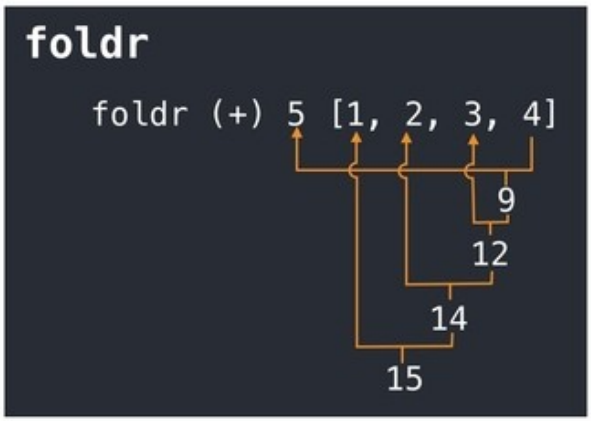
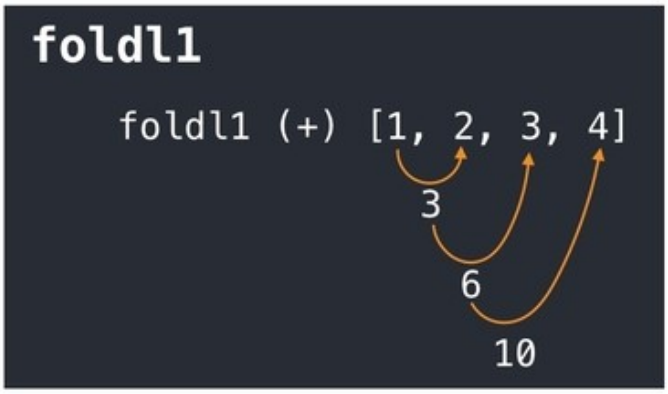
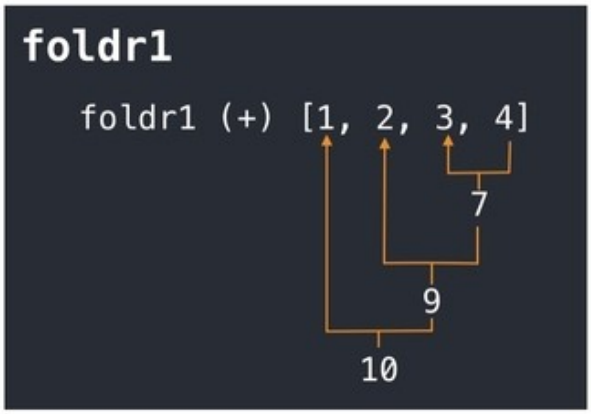
Ejercicio foldl, foldr

foldr :: (a->b->b) -> b -> [a] -> b
foldr f e [] = e
foldr f e (x:xs) = f x (foldr f e xs)









foldl :: (a->b->b) -> b -> [a] -> b
foldl f e [] = e
foldr f e (x:xs) = foldl f (f e x) xs

foldl (-) 0 [1,2,3,4] /= foldr (-) 0 [1,2,3,4]

Ejercicio especificar y derivar
g: [Num] → Bool
g.xs=<∃ as,bs : xs=as++bs : sum.as<0 >
.
.
.
generalizamos
gg: [Num] → Num → Bool
gg.[].n = n<0
gg.(x:xs).n= n<0 v gg.xs.(n+x)

g: [Num] → Bool
g.xs = gg.xs.0

Ejercicio función en Haskell
f::Int → [Int] → Bool
f _ [] = False
f 0 _ = False
f n (x:xs) = (nApariciones(take n (x:xs)) x)>1
                || f(n-1) xs

nApariciones :: [Int] → Int → Int
nApariciones [] _ = 0
nApariciones (x:xs) y | x==y = 1 + nApariciones xs y
                      | otherwise = nApariciones xs y