



## EJERCICIO 5

### a) Suma los elementos de una lista de Integer

sumaList :: [Integer] -> Integer

sumaList [] = 0

sumaList [a] = a

sumaList (a:xs) = a + sumaList (xs)

sumaList2 :: [N] -> N

sumaList2 [] = 0

sumaList2 [a] = a

sumaList2 (a:xs) = suma a (sumaList2 (xs))

### b) Multiplica los elementos de una lista de Integer

multiList :: [Integer] -> Integer

multiList [] = 0

multiList [a] = a

multiList (a:xs) = (a) \* multiList (xs)

multiList2 :: [N] -> N

multiList2 [] = 0

multiList2 [a] = a

multiList2 (a:xs) = producto a (multiList2 (xs))

### c) Invierte el orden de los elementos de una lista de N

invertir :: [Integer] -> [Integer]

invertir [] = []

invertir [a] = [a]

invertir (a:xs) = concatenar (invertir (xs)) [a]

### d) Devuelve los primeros n términos de una lista

tomar\_n :: Integer -> [Integer] -> [Integer]

tomar\_n 0 (a) = []

tomar\_n n [] = []

tomar\_n n (a:xs) = a : tomar\_n (n-1) (xs)

### e) Elimina los primeros n términos de una lista

borrar\_n :: Integer -> [Integer] -> [Integer]

borrar\_n 0 (a) = (a)

borrar\_n n [] = []

borrar\_n n (a:xs) = borrar\_n (n-1) (xs)

### Extra:

saca3 :: [Integer] -> [Integer]

saca3 [] = []

saca3 (a:xs) = **if** (a==3) **then** saca3 (xs) **else**

a : saca3 (xs)

### f) Devuelve la posición de un elemento en una lista

posicion :: [Integer] -> Integer -> Integer

posicion [ ] n = 0

posicion (a:xs) n = | a==n = 1

| otherwise = posicion (xs) n + 1

### g) mapear Devuelve las imágenes de una lista de enteros según una función

factorial :: Int -> Int

factorial 0 = 1

factorial x = x \* factorial (x-1)

mapear :: (Int -> Int) -> [Int] -> [Int]

mapear funcion [] = []

mapear **funcion** (a:xs) = (**funcion** a): mapear **funcion** (xs)

mapear factorial [] = []

mapear factorial (a:xs) = (factorial a): mapear factorial (xs)

**h) filtrar** Devuelve una lista para los cuales la función dada es Verdadera

```
filtrar :: (Int -> Bool) -> [Int] -> [Int]
filtrar funcion [] = []
filtrar funcion (a:xs) = if (funcion a == True) then a : filtrar funcion (xs) else filtrar funcion (xs)
```

**i) existe** Devuelve True si un valor de la lista verifica la función dada

```
existe :: [Int] -> (Int -> Bool) -> Bool
existe [] funcion = False
existe (a:xs) funcion = if (funcion a == True) then True else existe (xs) funcion
```

**j) pertenece** Devuelve True si un elemento esta en una lista

```
pertenece :: Integer -> [Integer] -> Bool
pertenece x [] = False
pertenece x (a : ys) | x==a = True
                    | otherwise = pertenece x (ys)
```

**k) borrar** Devuelve una lista dada sin un elemento especificado

```
borrar::[Int]-> Int-> [Int]
borrar [] b = []
borrar (a:xs) b = | b==a = borrar (xs) b
                 | otherwise = a : borrar (xs) b
```

**l) insertaOrd** Inserta un elemento en una lista ordenada

```
insertaOrd :: [Int] -> Int -> [Int]
insertaOrd [] x = [x]
insertaOrd (a:ys) x = | x<a) = x : (a:ys)
                    | otherwise = a : insertaOrd (ys) x
```

**m) multiplicar** Multiplica por un natural toda la lista de Naturales

```
multiplicarList::[Integer]->Integer ->Integer
multiplicarList [] n= []
multiplicarList (a:xs) n = n*a : multiplicarList (xs) n
```

**n) capicua** Determina si una lista es o no capicúa

Preguntar al Profesor  
-- Con Invertir --

## EJERCICIOS COMPLEMENTARIOS

### EJERCICIO 7 [Devuelve el elemento mayor de una lista](#)

```
mayor :: [Int] -> Int
mayor [] = error "No hay elementos en la lista"
mayor [a] = a
mayor (a:b:xs) = if (a>b) then mayor (a:xs) else mayor (b:xs)
```

### EJERCICIO 8 [Devuelve los primeros elementos de una lista de pares ordenados](#)

```
primeros_elem :: [(Integer,Integer)]->[Integer]
primeros_elem [] = []
primeros_elem [(a,b)] = [a]
primeros_elem ((a,b):xs) = a : primeros_elem (xs)
```

### EJERCICIO 9 [Devuelve los elementos mayores de cada par en una lista de pares](#)

```
mayores :: [(Integer,Integer)]->[Integer]
mayores [] = []
mayores [(a,b)] = if (a>b) then [a] else [b]
mayores ((a,b):ys) = if (a>b) then a : mayores (ys) else b : mayores (ys)
```

### EJERCICIO 10 [Devuelve los elementos pares de una lista](#)

```
pares :: [Integer]->[Integer]
pares [] = []
pares (a : xs) = if (mod a 2 == 0) then a : pares (xs) else pares (xs)
```

```
pares :: [N]->[N]
pares [] = []
pares (a : xs) = if (espar a = True) then a : pares (xs) else pares (xs)
```

### EJERCICIO 11 [Devuelve la suma de los elementos impares de una lista](#)

```
impares :: [Integer]->Integer
impares [] = 0
impares (a : xs) = if (mod a 2 /= 0) then a + impares(xs) else impares(xs)
```

```
impares :: [N]->N
impares [] = 0
impares (a : xs) = if (esimpar a = True) then (suma a) impares(xs) else impares(xs)
```

### EJERCICIO 12 [Devuelve el elemento del lugar n de una lista](#)

```
devolver :: Integer -> [Integer] -> Integer
devolver n [] = error "No hay elementos"
devolver n (a:xs) = if (n > largo (a:xs)) then error "No hay un elemento que ocupe ese lugar"
                    else if (n == 0) then a else devolver (n-1) (xs)
```

**EJERCICIO 13** Devuelve el menor múltiplo de 3 de una lista

```
menor :: [Int] -> Int
menor [] = 0
menor [a] = a
menor (a: b: xs) = if (a<b) then menor (a:xs) else menor (b:xs)
```

```
todosMultiplo3 :: [Int] -> [Int]
todosMultiplo3 [] = error "No hay elementos en la lista"
todosMultiplo3 [a] = if (mod a 3 == 0) then [a] else []
todosMultiplo3 (a: xs) = if (mod a 3 == 0) then a : todosMultiplo3 (xs) else todosMultiplo3 (xs)
```

```
menorMultiplo3 :: [Int] -> Int
menorMultiplo3 (xs) = menor (todosMultiplo3 (xs))
```

**EJERCICIO 14** Devuelve una lista sin elementos repetidos

```
esta :: Integer -> [Integer] -> Bool
esta x [] = False
esta x (a : ys) = if (x==a) then True else esta x (ys)
```

```
sinRepetir :: [Int] -> [Int]
sinRepetir [] = []
sinRepetir [a] = [a]
sinRepetir (a:xs) = if (esta a (xs) == True) then sinRepetir (xs) else a: sinRepetir (xs)
```

**EJERCICIO 15** Indica si un String tiene una vocal

```
contieneVocal :: String -> Bool
contieneVocal [] = False
contieneVocal (a:xs) = if (a=='a' || a=='e' || a=='i' || a=='o' || a=='u') then True else contieneVocal (xs)
```

**EJERCICIO 16** Elimina un carácter de un String

```
eliminar :: [char]-> String -> String
eliminar c [] = error "Falta String"
eliminar [] x = error "Falta Char"
eliminar c (a:xs) = if (c==a) then eliminar c (xs) else a : eliminar c (xs)
```

**EJERCICIO 17** Devuelve los divisores de un número natural

```
generaDiv :: Int -> Int -> [Int]
generaDiv n 1 = [1]
generaDiv n d = if (mod n d == 0) then d : generaDiv n (d - 1) else generaDiv n (d - 1)
```

```
divisores :: Int -> [Int]
divisores n = generaDiv n n
```

**EJERCICIO 18** Devuelve una lista en forma ordenada de menor a mayor

```
insertOrd :: Int -> [Int] -> [Int]
insertOrd x [] = [x]
insertOrd x (a:ys) = if (x<=a) then x : (a:ys) else a : insertOrd x (ys)
```

```
ordena :: [Int] -> [Int]
ordena [] = []
ordena (a : xs) = insertOrd a (ordena xs)
```

**EJERCICIO 19** Devuelve una lista que es concatenación de otras listas

```
concatenaListas :: [[a]] -> [a]
concatenaListas [] = []
concatenaListas (xs:xss) = xs ++ concatenaListas xss
```

**EJERCICIO 20** Devuelve el resultado de aplicar una función binaria a dos listas

```
prod :: Int -> Int -> Int
prod a b = a*b

funProducto :: [Int] -> [Int] -> [Int]
funProducto [] (n) = []
funProducto (n) [] = []
funProducto (a:xs) (b:ys) = (prod a b) : funProducto (xs) (ys)
```

**EJERCICIO 21** Devuelve una lista de pares a partir de dos listas

```
funPares :: [Int] -> [Int] -> [(Int,Int)]
funPares [] [] = []
funPares [] (a : xs) = []
funPares (a : xs) [] = []
funPares (a : xs) (b : ys) = (a,b) : funPares (xs) (ys)
```

```
forPares :: Ord a => [a] -> [b] -> [(a,b)]
forPares [] [] = []
forPares [] (a:xs) = []
forPares (a : xs) [] = []
forPares (a : xs)(b : ys)=(a,b) : forPares (xs)(ys)
```