EJERCICIOS VISTOS EN CLASE

1) Siguiente de un número.

```
sig :: Int -> Int
sig x = x + 1
```

2) Doble de un número.

```
doble :: Int -> Int
doble x = 2 * x
```

3) Intercambiar valores.

```
swap :: (Int, Int) -> (Int, Int)
swap (x, y) = (y, x)
```

4) Devuelve el valor absoluto de un número.

```
absoluto :: Int -> Int
absoluto n = if (n > 0) then n else (-n)
```

5) Devuelve el factorial de un número.

```
fact :: Int -> Int
fact 0 = 1
fact n = n * fact(n-1)
```

6) Producto de los números de una lista.

```
producto :: [Int] -> Int
producto [] = 1
producto (x:y) = x * producto y
```

7) Devuelve el máximo valor de una lista.

```
maxLista :: [Int] -> Int
maxLista (x:[]) = x
maxLista (x : (y:z)) = if x < y then maxLista (y:z) else maxLista (x:z)

maxLista2 :: [Int] -> Int
maxLista2 (x:[]) = x
maxLista2 (x:y) = if (x > maxLista2 y) then x else maxLista2 y
```

8) Cuantas veces está el número ingresado en la lista.

```
cuantas :: (Int, [Int]) -> Int
cuantas (n, []) = 0
cuantas (n, (x:y)) = if n == x then 1 + cuantas (n, y) else cuantas (n, y)
```

9) Reemplazar un número dado por otro en una lista.

```
buscoyReemplazo :: (Int, Int, [Int]) -> [Int]
buscoyReemplazo (a, b, []) = []
buscoyReemplazo (a, b, (x:y)) = if a == x then b : buscoyReemplazo (a, b, y) else x : buscoyReemplazo (a, b, y)
```

10) Merge de 2 listas (genera una única lista ordenada a partir de 2 listas).

```
merge :: ([Int], [Int]) -> [Int]
merge (z, []) = z
merge ([], z) = z
merge (x:xs, y:ys) = if x < y then x : (merge (xs, y:ys)) else y : (merge (x:xs, ys))
```

11) Merge de 3 listas (genera una única lista ordenada a partir de 3 listas).

```
merge3 :: ([Int], [Int], [Int]) -> [Int]
merge3 (a, b, c) = merge(merge(a, b), c)
```

12) Sumar todos los elementos (Int) de una lista de listas.

```
sumaListas :: [[Int]] -> Int
sumaListas [] = 0
sumaListas ([]:y) = sumaListas y
sumaListas ((x:y):z) = x + sumaListas (y:z)
```

13) Concatenar listas de listas.

```
appendListas :: [[a]] -> [a]
appendListas [] = []
appendListas (x:y) = append (x, appendListas y)

append :: ([a], [a]) -> [a]
append ([], z) = z
append ((x:y), z) = x : append (y, z)
```