PRÁCTICO NÚMERO 3 – PROGRAMACIÓN DECLARATIVA

VENTURA GINO

EJERCICIO 1:

Redefinir la función map de tal forma que map f lista-> devuelva la lista obtenida aplicando f a cada elemento de lista. Hacer dos versiones de la función:

- Aplicando recursividad.
- Aplicando listas intensionales.

CÓDIGO:

- **Version 1:**

```
map1 :: (a  -> b)  -> [a]  -> [b]

map1 f [] = []

map1 f (x:xs) = f x : map2 f xs
```

- **Version 2:**

```
map2 f xs = [f x | x  <- xs]
```

```
*Main> map1 (3*) [1,2,3]
[3,6,9]
*Main> map2 (3*) [1,2,3]
[3,6,9]
```

EJERCICIO 2:

Redefinir la función sum de tal forma que sum de lista -> devuelva la suma de los elementos de lista.

Ejemplo: sum' [1,3,6] -> 10

CÓDIGO:

```
sumar [] = 0

sumar (x:xs) = x + sumar xs
```

```
Prelude> :load "Practico3-Ejercicio2.hs"
[1 of 1] Compiling Main             ( Practico3-Ejercicio2.hs, interpreted )
Ok, one module loaded.
*Main> sumar [1,3,6]
10
```

EJERCICIO 3:

Definir una función que tome dos listas xs ys y verifica si las dos listas xs e ys son iguales, devuelve True si son iguales sino False.

CÓDIGO:

```
comprobarListas xs xy | xs == xy = True
      | otherwise = False
```

```
[1 of 1] Compiling Main             ( Practico3-Ejercicio3.hs, interpreted )
Ok, one module loaded.
*Main> comprobarListas [1,2] [1,2]
True
*Main> comprobarListas [1,2] [1,1]
False
```

EJERCICIO 4:

Definir una función tal forma que sume de los cuadrados de los elementos de la lista l.

Por ejemplo: sum_cuadrados [1,2,3] -> 14

CÓDIGO:

```
cuadrado [] = 0

cuadrado (x:xs) = x*x + cuadrado xs
```

```
Prelude> :load "Practico3-Ejercicio4.hs"
[1 of 1] Compiling Main             ( Practico3-Ejercicio4.hs, interpreted )
Ok, one module loaded.
*Main> cuadrado [1,2,3]
14
```

EJERCICIO 5:

Redefinir la función filter de tal forma que filter' p lista -> devuelva la lista de los elementos de lista que cumplen la propiedad p. Por ejemplo,

filter even [1,2,3,4,5,6,7] -> [2,4,6]

filter (>3) [1,2,3,5,6,7] -> [5,6,7]

CÓDIGO:

```
filtro :: (x -> Bool) -> [x] -> [x]
filtro p [] = []
filtro p (x:xs) | p x = x : filtro p xs
     | otherwise = filtro p xs
```

```
[1 of 1] Compiling Main             ( Practico3-Ejercicio5.hs, interpreted )
Ok, one module loaded.
*Main> filtro (3>)[1,2,3,4,5,6,7]
[1,2]
```

EJERCICIO 6:

Redefinir la función head de tal forma que head' lista es el head de lista. Por ejemplo:

head [3,5,2] -> 3

head [] -> "Error lista vacia"

CÓDIGO:

```
primero :: [Int] -> Int
primero [] = error "Error, la lista está vacía."
primero (x:xs) = x
```

```
[1 of 1] Compiling Main             ( Practico3-Ejercicio6.hs, interpreted )
Ok, one module loaded.
*Main> primero [3,4,5]
3
*Main> primero []
*** Exception: Error, la lista está vacía.
CallStack (from HasCallStack):
  error, called at Practico3-Ejercicio6.hs:2:14 in main:Main
```

EJERCICIO 7:

Definir una función que reciba una lista como parámetro y devuelta true si la lista tiene valores repetidos y false en caso contrario

Ejemplo:

duplicados [1,2,3,4] -> false

duplicados [1,2,3,2] -> true

CÓDIGO:

```
import Data.List

distinto :: Eq a => [a] -> Bool

distinto x = length  x == length (nub x)
```

```
[1 of 1] Compiling Main             ( Practico3-Ejercicio7.hs, interpreted )
Ok, one module loaded.
*Main> distinto [1,2,3,4,5]
True
*Main> distinto [1,2,1,2,1]
False
```

EJERCICIO 8:

Definir una función que reciba una lista y valide si la lista esta ordenada de menor a mayor, si lo cumple devuelve true sino false

CÓDIGO:

```
listaOrd :: [Int] -> Bool

listaOrd [] = True

listaOrd [_] = True

listaOrd (x:y:xs) = (x <= y) && listaOrd (y:xs)
```

```
[1 of 1] Compiling Main             ( Practico3-Ejercicio8.hs, interpreted )
Ok, one module loaded.
*Main> listaOrd [1,2,3,4,5]
True
*Main> listaOrd [1,6,4,2,3]
False
```

EJERCICIO 9:

Definir una función que calcule el valor mayor de tres pasados como parametros. Por ejemplo:

mayor 6 2 4 == 6

mayor 6 7 4 == 7

mayor 6 7 9 == 9

CÓDIGO:

```
mayor x y z | x > y && x > z = x
 |y > x && y > z = y
 |z > x && z > y = z
 |otherwise = error "No cumple ninguna condición"
```

```
Prelude> :load "Practico3-Ejercicio9.hs"
[1 of 1] Compiling Main             ( Practico3-Ejercicio9.hs, interpreted )
Ok, one module loaded.
*Main> mayor 6 2 4
6
*Main> mayor 6 7 4
7
*Main> mayor 6 7 9
9
*Main> mayor 4 4 4
*** Exception: No cumple ninguna condición
CallStack (from HasCallStack):
  error, called at Practico3-Ejercicio9.hs:4:15 in main:Main
```

EJERCICIO 10:

Definir una función que reciba una lista como parámetro y devuelva otra lista de tal forma que el primer elemento de la lista sea el último en la lista resultante. Por ejemplo:

cambiar [1,2,3] -> [2,3,1]

CÓDIGO:

No lo pude Resolver.