1. Utilizá las definiciones intuitivas de los operadores de listas para evaluar las siguientes expresiones. Subrayá la subexpresión resuelta en cada paso justificado. Luego usá un intérprete de `haskell` para verificar los resultados. Por ejemplo:

$head.xs$	`head xs`
$tail.xs$	`tail xs`
$x \triangleright xs$	`x:xs`
$xs \triangleleft x$	`xs ++ [x]`
$xs \uparrow n$	`take n xs`
$xs \downarrow n$	`drop n xs`
$xs +\!\!+ ys$	`xs ++ ys`
$\#xs$	`length xs`
$xs \,!\, n$	`xs !! n`

$a)$ $\#[5,6,7]$

$b)$ $[5,3,57] \,!\, 1$

$c)$ $[0,11,2,5] \triangleright [\,]$

$d)$ $[5,6,7] \uparrow 2$

$e)$ $[5,6,7] \downarrow 2$

$f)$ $head.(0 \triangleright [1,2,3])$

a)
```
length [5,6,7]
= {def. de #}
3
```

b)
```
[5,3,57] ! 1
= {def. de !}
3
```

c)
```
[0,11,2,5]:[]
= {def. :}
[0,11,2,5]
```

d)
```
take 2 [5,6,7]
= {def. de take}
[5,6]
```

e)
```
drop 2 [5,6,7]
= {def. de drop}
[7]
```

f)
```
head (0:[1,2,3])
= {def. de :}
head [0,1,2,3]
= {def. de head}
0
```

2. Decidí si las siguientes expresiones están bien escritas, agregando paréntesis para hacer explícita la precedencia y la asociatividad. Usá un intérprete de `haskell` para verificar los resultados.

$a)$ $-45 \triangleright [1,2,3]$

$b)$ $([1,2] +\!\!+ [3,4]) \triangleleft 5$

$c)$ $0 \triangleleft [1,2,3]$

$d)$ $[\,] \triangleright [\,]$

```
(-45):[1,2,3]
```

```
([1,2] ++ [3,4]) ++ [5] == [1,2] ++ [3,4] ++ [5]
```

```
[0] ++ [1,2,3]
```

```
[]:[]
```

e) $([1] +\!\!+ [2]) \triangleleft [3]$

```
([1] ++ [2]) ++ [3] == [1] ++ [2] ++ [3]
```

f) $[1, 5, False]$

No tipa

g) $head.[[5]]$

```
head [[5]]
```

h) $head.[True, False] +\!\!+ [False]$

```
(head [True, False]) ++ [False]
```

3. Una función de **filter** es aquella que dada una lista devuelve otra lista cuyos elementos son los elementos de la primera que cumplan una determinada condición, en el mismo orden y con las mismas repeticiones (si las hubiere). Por ejemplo: $soloPares : [Int] \to [Int]$ devuelve aquellos elementos de la lista que son pares.

   Definí recursivamente las siguientes funciones filter.

a) $soloPares : [Int] \to [Int]$, que dada una lista de enteros xs devuelve una lista sólo con los números pares contenidos en xs, en el mismo orden y con las mismas repeticiones (si las hubiera).
   Por ejemplo: $soloPares.[3, 0, -2, 12] = [0, -2, 12]$

```
soloPares :: [Int] -> [Int]
soloPares ls
    | null ls = []
    | even cabeza = [cabeza] ++ soloPares cola
    | otherwise = soloPares cola
    where
    cabeza = head ls
    cola = tail ls
```

b) $mayoresQue10 : [Int] \to [Int]$, que dada una lista de enteros xs devuelve una lista sólo con los números mayores que 10 contenidos en xs,
   Por ejemplo: $mayoresQue10.[3, 0, -2, 12] = [12]$

```
mayoresQue10 :: [Int] -> [Int]
mayoresQue10 ls
    | null ls = []
    | cabeza > 10 = [cabeza] ++ mayoresQue10 cola
    | otherwise = mayoresQue10 cola
    where
    cabeza = head ls
    cola = tail ls
```

c) *mayoresQue* : $Int \to [Int] \to [Int]$, que dado un entero n y una lista de enteros xs devuelve una lista sólo con los números mayores que n contenidos en xs,
Por ejemplo: *mayoresQue*,2.$[3, 0, -2, 12] = [3, 12]$

```
mayoresQue :: Int -> [Int] -> [Int]
mayoresQue n ls
    | null ls = []
    | cabeza > n = [cabeza] ++ mayoresQue n cola
    | otherwise = mayoresQue n cola
    where
    cabeza = head ls
    cola = tail ls
```

4. Una función de **map** es aquella que dada una lista devuelve otra lista cuyos elementos son los que se obtienen de aplicar una función a cada elemento de la primera en el mismo orden y con las mismas repeticiones (si las hubiere). Por ejemplo: *duplica* : $[Int] \to [Int]$ devuelve cada elemento de la lista multiplicado por 2.

Definí recursivamente las siguientes funciones de map.

a) *sumar1* : $[Int] \to [Int]$, que dada una lista de enteros le suma uno a cada uno de sus elementos.
Por ejemplo: *sumar1*.$[3, 0, -2] = [4, 1, -1]$

```
sumar1 ls
    | null ls = []
    | otherwise = [helper cabeza] ++ sumar1 cola
    where
    cabeza = head ls
    cola = tail ls
    helper = (+1)
```

b) *duplica* : $[Int] \to [Int]$, que dada una lista de enteros duplica cada uno de sus elementos.
Por ejemplo: *duplica*.$[3, 0, -2] = [6, 0, -4]$

```
duplica ls
    | null ls = []
    | otherwise = [helper cabeza] ++ duplica cola
    where
    cabeza = head ls
    cola = tail ls
    helper = (*2)
```

c) *multiplica* : $Int \to [Int] \to [Int]$, que dado un número n y una lista, multiplica cada uno de los elementos por n.
Por ejemplo: *multiplica*,3.$[3, 0, -2] = [9, 0, -6]$

```
multiplica n ls
    | null ls = []
    | otherwise = [helper cabeza] ++ multiplica n cola
    where
    cabeza = head ls
    cola = tail ls
    helper = (*n)
```

5. Una función de **fold** es aquella que dada una lista devuelve un valor resultante de combinar los elementos de la lista. Por ejemplo: *sum* : $[Int] \to Int$ devuelve la sumatoria de los elementos de la lista.
Definí recursivamente las siguientes funciones fold.

a) *todosMenores10* : $[Int] \to Bool$, que dada una lista devuelve *True* si ésta consiste sólo de números menores que 10.

```
todosMenores10 ls
    | null ls = True
    | otherwise = (cabeza < 10) && todosMenores10 cola
    where
    cabeza = head ls
    cola = tail ls
```

b) *hay0* : $[Int] \to Bool$, que dada una lista decide si existe algún 0 en ella.

```
hay0 ls
    | null ls = False
    | otherwise = (cabeza == 0) || hay0 cola
    where
    cabeza = head ls
    cola = tail l
```

$c)$ $sum : [Int] \to Int$, que dada una lista devuelve la suma de todos sus elementos.

```
suma ls
    | null ls = 0
    | otherwise = cabeza + (sum cola)
    where
    cabeza = head ls
    cola = tail ls
```