

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:

	haskell	resultado
a) <u>#</u> [5,6,7]	length [5,6,7]	3
b) [5,3,57] <u>!</u> 1	[5,3,57] !! 1	3
c) [0,11,2,5] <u>▷</u> []	[0,11,2,5] : []	[[0,11,2,5]]
d) [5,6,7] <u>↑</u> 2	take 2 [5,6,7]	[5,6]
e) [5,6,7] <u>↓</u> 2	drop 2 [5,6,7]	[7]
f) head.(0 <u>▷</u> [1,2,3])	head (0:[1,2,3])	0

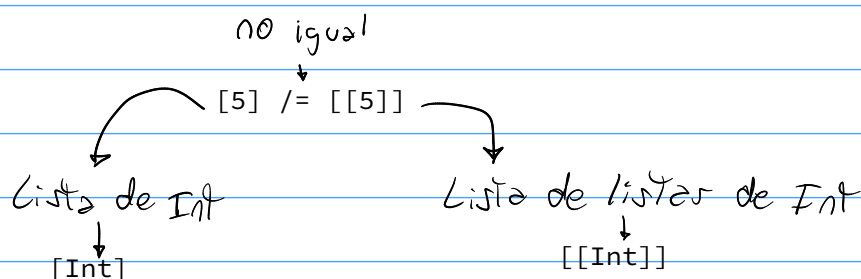
En **haskell** los distintos operadores se pueden escribir así:

<i>head.xs</i>	head xs
<i>tail.xs</i>	tail xs
<i>x▷xs</i>	x:xs
<i>xs◁x</i>	xs ++ [x]
<i>xs↑n</i>	take n xs
<i>xs↓n</i>	drop n xs
<i>xs++ys</i>	xs ++ ys
<i>#xs</i>	length xs
<i>xs!n</i>	xs !! n

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) <u>-</u> 45▷[1,2,3]	(-45):[1,2,3]	✓
b) ([1,2] ++ [3,4]) ◁ 5	[1,2]++[3,4]++[5]	✓
c) 0 ◁ [1,2,3]	[0] ++ [1,2,3]	✓
d) [] ▷ []	[] : []	✓

e) ([1] ++ [2]) ◁ [3]	[1]++[2]++[3]	✓
f) [1,5, False]	[(1,5,False)]	✓
g) head.[[5]]	head [[5]]	✓
h) head.[True, False] ++ [False]	head ([True, False] ++ [False])	✓



a) $\text{soloPares} : [Int] \rightarrow [Int]$, 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: $\text{soloPares}.[3, 0, -2, 12] = [0, -2, 12]$

1. Definir el caso base

2. Ver que operación realizar en cada elemento

Tener en cuenta: Llamar a la función con una lista más pequeña que la original
Separar la cabeza de la cola.

1. $\text{soloPares} [] = []$

2. Ver si cumple (even n), de ser así mantener el elemento en la lista

Si no se cumple (even n), quitar el elemento

} caso
recursivo

```
soloPares [] = []  
soloPares (x:xs)  
  | even x = [x] ++ soloPares xs  
  | otherwise = soloPares xs
```

Evaluamos:

$1:[2,3] == [1,2,3]$

```
soloPares 1:[2,3]  
  | even 1 = [1] ++ soloPares [2,3]  
  | otherwise = soloPares [2,3]           [] ++ [2]
```

```
soloPares 2:[3]  
  | even 2 = [2] ++ soloPares [3]           [2] ++ []  
  | otherwise = soloPares [3]
```

```
soloPares 3:[]  
  | even 3 = [3] ++ soloPares []           []  
  | otherwise = soloPares []
```

Si juntamos todo:

`[] ++ [2] ++ []`

eso es lo que evalúa Haskell