



## Ejercicios para resolver en papel

### 5.2 Patrones




**ATENCIÓN:** para resolver los ejercicios de esta práctica tenga en cuenta lo siguiente.



USE SOLO `foldr`, `filter` y/o `map` para resolver.

Si define **funciones auxiliares**, **NO** puede usar **recursión simple**, use patrones.



En **NINGÚN CASO** use funciones predefinidas para manipular listas, como `length`, `member`, `append`, etc.


### Sistema de contraseñas

Considerando los ejercicios del **Sistema de contraseñas** enunciados en **Ejercicios para resolver en papel - 5.1 Estructuras y Listas**, resuelva utilizando patrones  **Ejercicio. 1**,  **Ejercicio. 2** y  **Ejercicio. 3** de esta práctica.

 **Ejercicio 1.** De dos nuevas definiciones de la función `cantAdmin` (5.1.  **Ejercicio. 4**) de acuerdo a lo siguiente.


- a) `cantAdmin` recorre más de una vez la lista recibida.
- \*b) `cantAdmin` recorre una única vez la lista recibida.

 **Ejercicio 2.** De una nueva definición de la función `bloquearClaves` (5.1.  **Ejercicio. 6**).

 **Ejercicio 3.** Complete el diseño de la función `idLargos` dando su definición. *Puede usar `string-length` o cualquier función predefinida que manipule `String`.*

```
;idLargos: List(Usr) ->List(String)
;Esta función recibe una lista de Usr y devuelve la lista de los identificadores
con más de 3 caracteres.
(check-expect (idLargos (list ANA LUIS MARTA)) (list "luis" "marta"))
(check-expect (idLargos (list ANA)) empty)
```

### Agregando un poco de complejidad


 **Ejercicio 4.** Complete el diseño de la función `sumNumericos` dando su definición. *Puede usar la función predefinida `string-numeric?`.*

```
;sumNumericos: List(String) ->Number
;Recibe una lista de Strings y devuelve la suma todos aquellos que son enteros no
negativos.
(check-expect (sumNumericos (list "12-9" "12" "sol" "1nos" "33.5" "10")) 22)
(check-expect (sumNumericos (list "-12" "sol" "33.5")) 0)
(check-expect (sumNumericos empty) 0)
```


---

 **Ejercicio 5.** Complete el diseño de la siguiente función dando su definición. *Use local.*

```
;infAn: List(Number) Number ->List(Number)
;infAn recibe una lista de números y un número n; y devuelve la lista de números
inferiores a n.
(check-expect (infAn (list 1 2 3 4) 3) (list 1 2))
(check-expect (infAn (list 1 2 3 4) 1) empty)
```

 **Ejercicio 6\*.** Complete el diseño de la función `minimo`, dando su definición. *Puede usar la función `min` predefinida en Racket .*

```
;minimo: List(Number) ->Number
;Esta función recibe una lista NO vacía de números cualesquiera y devuelve el menor.
(check-expect (minimo (list -1.6 5 3 -80 6 57.9 0)) -80)
```

 **Ejercicio 7\*.** Complete el diseño de la función `primeroPares`, dando su definición.

```
;primeroPares: List(Number) ->List(Number)
;Esta función recibe una lista NO vacía de enteros y los ordena ubicando primero
los pares y luego los impares.
(check-expect (primeroPares (list 9 -5 6 3 -2 8 0 12 100 7))
              (list 6 -2 8 0 12 100 9 -5 3 7))
```