Auxiliar 2
Map, filter, foldl y curry

P1. *Orden Superior:* Programe su propia versión de las siguientes funciones.

(a) MyMap, función que recibe una función y una lista y aplica la función recibida a todos los elementos de la lista, retornando una lista.

(b) MyFoldl, función que recibe una función de dos argumentos, un valor inicial y una lista, y aplica la función a los elementos de la lista y el valor obtenido previamente.

(c) MyFilter, función que recibe un predicado y una lista y retorna una lista con los elementos de la lista que cumplen el predicado.

P2. *Pongamos en práctica map, filter y foldl*

(a) Estás trabajando para el INE en el censo, y se tienen datos de las edades de las personas en una lista, pero vienen con datos basura, con nulls escritos como f, además las edades vienen en formato string y no en número, es decir se tiene una fila como la siguiente:

```
(list "10" #f "40" "10" #f)
```

Se quiere calcular el promedio de las edades, en el caso anterior sería

```
(/ (+ 10 10 40) 3)
<< 20
```

Programa la función **prom_edad** que calcule este promedio, lance un error si la lista solo contiene nulls. Use las funciones map, filter y fold

(b) Ahora la lista de datos no solo viene con los datos de las edades y los f, si no que también un elemento mismo de la lista puede ser otra lista que contenga los mismos elementos, es decir, es una estructura recursiva.

Para darle un nombre a los datos, serian del tipo A:

```
;; define A = ListOf[Sting/Boolean/A]
```

Entonces un ejemplo sería

```
(list "10" (list "10" #f "40" "10" #f) "40" "10" #f)
```

Programe la función **calcular_prom_rec** que calcule el promedio de edades, sabiendo que hay datos recursivos.

P3. *Currificación de funciones*

(a) Defina la función `(curry-n n f)` que recibe una función de n argumentos y retorna su versión currificada.

(b) Defina la función `(uncurry-2 f)` que recibe una función currificada de dos argumentos y devuelve su versión no currificada.