

Funcional 4: Listas + Orden Superior

Listas

Existen funciones predefinidas en el Prelude que nos permiten manejar listas por ej: head / tail / !! (infija, devuelve el elemento en la posición i, base 0). Ejemplos

```
head [2,4,6,8] = 2
tail [2,4,6,8] = [4,6,8]
[2,4,6,8] !! 1 = 4    -- base 0!!
null [] = True    --Indica si una lista está vacía.
null [2,4,5] = False
concat [[1..4],[11..13],[21,34]] = [1,2,3,4,11,12,13,21,34] --
"aplana" una lista de listas.
```

Si se quiere calcular el promedio, dada una lista números y se está usando prelude puro, se puede hacer algo así:

```
> sum [3,5,6] / fromInteger(toInteger(length[3,5,6]))
4.666666666666667
```

Tener en cuenta que en la notación [a..b] ni a ni b tienen por qué ser constantes, pueden ser cualquier expresión, p.ej

```
[1..head [6,3,8]]           [min (3+4) (3*4)..max (3+4) (3*4)]
```

Ejercicio Extra

Para cada función que definan, obtengan su tipo y después verifiquen con lo que les dice el :t. ¡Pero háganlo primero a mano! si no, no practican. Hacer lo mismo con las funciones head, tail y !!. Para preguntar por el tipo de una función infija, la ponen entre paréntesis, p.ej. :t (!!)

Ejercicios

1. Definir una función que sume una lista de números.
Nota: Investigar sum
2. Durante un entrenamiento físico de una hora, cada 10 minutos de entrenamiento se tomó la frecuencia cardíaca de uno de los participantes obteniéndose un total de 7 muestras que son las siguientes:
frecuenciaCardiaca = [80, 100, 120, 128, 130, 123, 125]
Comienza con un frecuencia de 80 min 0.
A los 10 min la frecuencia alcanza los 100

A los 20 min la frecuencia es de 120,
 A los 30 min la frecuencia es de 128
 A los 40 min la frecuencia es de 130, ...etc..
 A los 60 min la frecuencia es de 125 **frecuenciaCardiaca** es un función constante.

- a. Definir la función **promedioFrecuenciaCardiaca**, que devuelve el promedio de la frecuencia cardíaca.

```
Main> promedioFrecuenciaCardiaca
115.285714285714
```

- b. Definir la función **frecuenciaCardiacaMinuto/1**, que recibe m que es el minuto en el cual quiero conocer la frecuencia cardíaca, m puede ser a los 10, 20, 30 ,40,..hasta 60.

```
Main> frecuenciaCardiacaMomento 30
128
```

Ayuda: Vale definir una función auxiliar para conocer el número de muestra.

- c. Definir la función **frecuenciasHastaMomento/1**, devuelve el total de frecuencias que se obtuvieron hasta el minuto m.

```
Main> frecuenciasHastaMomento 30
[80, 100, 120, 128]
```

Ayuda: Utilizar la función take y la función auxiliar definida en el punto anterior.

- d.

3. Definir la función **esCapicua/1**, si data una lista de listas, me devuelve si la concatenación de las sublistas es una lista capicua..Ej:

```
Main> esCapicua ["ne", "uqu", "en"]
True
```

Porque "neuquen" es capicua.

Ayuda: Utilizar concat/1, reverse/1.

4. Se tiene información detallada de la duración en minutos de las llamadas que se llevaron a cabo en un período determinado, discriminadas en horario normal y horario reducido.

```
duracionLlamadas =
```

```
((("horarioReducido",[20,10,25,15]),(("horarioNormal",[10,5,8,2,9,10]))).
```

- a. Definir la función **cuandoHabloMasMinutos**, devuelve en que horario se habló más cantidad de minutos, en el de tarifa normal o en el reducido.

```
Main> cuandoHabloMasMinutos
"horarioReducido"
```

- b. Definir la función **cuandoHizoMasLlamadas**, devuelve en que franja horaria realizó más cantidad de llamadas, en el de tarifa normal o en el reducido.

```
Main> cuandoHizoMasLlamadas
```

```
"horarioNormal"
```

Nota: Utilizar composición en ambos casos

Orden Superior

Nota

Si una función f recibe en algunos de sus argumentos una función entonces f es una función de orden superior. Veamos un ejemplo.

Si quiero aplicar a un número n una función determinada podría hacer..

```
aplicar f n = f n
```

Por ejemplo, le paso como argumento una función aplicada parcialmente: $(+3)$, $(4*)$.

```
Main> aplicar (+ 3) 2
```

```
5
```

```
Main> aplicar (4 *) 3
```

```
12
```

Ejercicios

1. Definir la función **existsAny/2**, que dadas una función booleana y una tupla de tres elementos devuelve True si existe algún elemento de la tupla que haga verdadera la función.

```
Main> existsAny even (1,3,5)
```

```
False
```

```
Main> existsAny even (1,4,7)
```

```
True
```

```
porque even 4 da True
```

```
Main> existsAny (0>) (1,-3,7)
```

```
True
```

```
porque even -3 es negativo
```

2. Definir la función **mejor/3**, que recibe dos funciones y un número, y devuelve el resultado de la función que dé un valor más alto. P.ej.

```
Main> mejor cuadrado triple 1
```

```
3
```

(pues triple 1 = 3 > 1 = cuadrado 1)

```
Main> mejor cuadrado triple 5
```

```
25
```

(pues cuadrado 5 = 25 > 15 = triple 5)

Nota: No olvidar la función max.

3. Definir la función **aplicarPar/2**, que recibe una función y un par, y devuelve el par que resulta de aplicar la función a los elementos del par. P.ej.

```
Main> aplicarPar doble (3,12)
```

```
(6,24)
```

```
Main> aplicarPar even (3,12)
```

```
(False, True)
```

```
Main> aplicarPar (even . doble) (3,12)
```

```
(True, True)
```

4. Definir la función **parDeFns/3**, que recibe dos funciones y un valor, y devuelve un par ordenado que es el resultado de aplicar las dos funciones al valor. P.ej.

```
Main> parDeFns even doble 12
```

```
(True, 24)
```

Orden Superior + Listas

Además existen funciones de orden superior predefinidas que nos permiten trabajar con listas. Por ej:

Si quiero filtrar todos los elementos de una lista determinada que cumplen una determinada condición puedo utilizar filter.

```
paresEntre n1 n2 = filter even [n1..n2]
```

Otro Ejemplo de funciones de orden superior predefinidas que se utiliza mucho es el map Si quiero transformar una lista de elementos, puedo hacer:

```
sumarN n lista = map (+n) lista
```

Suma n a cada elemento de la lista.

```
sumarElDobleDeN n lista = map (+ (doble n)) lista
```

Aplica el doble a cada elemento de la lista.

Otras funciones de orden superior:

`all even [2,48,14] = True` -- Indica si todos los elementos de una lista cumplen una condición.

`all even [2,49,14] = False`

`any even [2,48,14] = True` -- Indica si algunos de los elementos de una lista cumplen una condición

Ejercicio Extra

Obtener :t el tipo de las funciones (4+) y (+4). Fíjense que en la segunda dice algo de "flip", los curiosos pueden investigar un poco más qué es eso.

Ejercicios

1. Definir la función **esMultiploDeAlguno/2**, que recibe un número y una lista y devuelve True si el número es múltiplo de alguno de los números de la lista.

P.ej.

```
Main> esMultiploDeAlguno 15 [2,3,4]
```

True,

porque 15 es múltiplo de 3

```
Main> esMultiploDeAlguno 34 [3,4,5]
```

False

porque 34 no es múltiplo de ninguno de los 3 Nota: Utilizar la función any/2.

2. Armar una función **promedios/1**, que dada una lista de listas me devuelve la lista de los promedios de cada lista-elemento. P.ej.

```
Main> promedios [[8,6],[7,9,4],[6,2,4],[9,6]]
```

```
[7,6.67,4,7.5]
```

Nota: Implementar una solución utilizando map/2.

3. Armar una función **promediosSinAplazos** que dada una lista de listas me devuelve la lista de los promedios de cada lista-elemento, excluyendo los que sean menores a 4 que no se cuentan. P.ej.

```
Main> promediosSinAplazos [[8,6],[6,2,6]]
```

```
[7,6]
```

Nota: Implementar una solución utilizando map/2.

4. Definir la función **mejoresNotas**, que dada la información de un curso devuelve la lista con la mejor nota de cada alumno. P.ej.

```
Main> mejoresNotas [[8,6,2,4],[7,9,4,5],[6,2,4,2],[9,6,7,10]]
```

```
[8,9,6,10].
```

Ayuda: Utilizar la función predefinida maximum/1.

5. Definir la función **aprobó/1**, que dada la lista de las notas de un alumno devuelve True si el alumno aprobó. Se dice que un alumno aprobó si todas sus notas son 6 o más. P.ej.

```
Main> aprobo [8,6,2,4]
```

```
False
```

```
Main> aprobo [7,9,6,8]
```

```
True
```

Ayuda: Utilizar la función predefinida `minimum/1`.

6. Definir la función **aprobaron/1**, que dada la información de un curso devuelve la información de los alumnos que aprobaron. P.ej.

```
Main> aprobaron [[8,6,2,4],[7,9,6,7],[6,2,4,2],[9,6,7,10]]
```

```
[[7,9,6,7],[9,6,7,10]]
```

Ayuda: usar la función `aprobó/1`.

7. Definir la función **divisores/1**, que recibe un número y devuelve la lista de divisores. P.ej.

```
Main> divisores 60
```

```
[1,2,3,4,5,6,10,12,15,20,30,60]
```

Ayuda: para calcular divisores `n` alcanza con revisar los números entre 1 y `n`.

8. Definir la función **exists/2**, que dadas una función booleana y una lista devuelve True si la función da True para algún elemento de la lista. P.ej.

```
Main> exists even [1,3,5]
```

```
False
```

```
Main> exists even [1,4,7]
```

```
True
```

porque `even 4` da True

9. Definir la función **hayAlgunNegativo/2**, que dada una lista de números y un (...algo...) devuelve True si hay algún nro. negativo.

```
Main> hayAlgunNegativo [2,-3,9] (...algo...)
```

```
True
```

10. Definir la función **aplicarFunciones/2**, que dadas una lista de funciones y un valor cualquiera, devuelve la lista del resultado de aplicar las funciones al valor. P.ej.

```
Main> aplicarFunciones[(*4),(+3),abs] (-8)
```

```
[-32,-5,8]
```

Si pongo:

```
Main> aplicarFunciones[(*4),even,abs] 8
```

da error. ¿Por qué?

11. Definir la función **sumaF/2**, que dadas una lista de funciones y un número, devuelve la suma del resultado de aplicar las funciones al número. P.ej.

```
Main> sumaF[(*4),(+3),abs] (-8)
-29
```

12. Un programador Haskell está haciendo las cuentas para un juego de fútbol virtual (como el Hattrick o el ManagerZone). En un momento le llega la información sobre la habilidad de cada jugador de un equipo, que es un número entre 0 y 12, y la orden de subir la forma de todos los jugadores en un número entero; p.ej., subirle 2 la forma a cada jugador. Ahora, ningún jugador puede tener más de 12 de habilidad; si un jugador tiene 11 y la orden es subir 2, pasa a 12, no a 13; si estaba en 12 se queda en 12. Escribir una función **subirHabilidad/2** que reciba un número (que se supone positivo sin validar) y una lista de números, y le suba la habilidad a cada jugador cuidando que ninguno se pase de 12. P.ej.

```
Main> subirHabilidad 2 [3,6,9,10,11,12]
[5,8,11,12,12,12]
```

13. Ahora el requerimiento es más genérico: hay que cambiar la habilidad de cada jugador según una función que recibe la vieja habilidad y devuelve la nueva. Armar: una función **flimitada** que recibe una función f y un número n , y devuelve $f\ n$ garantizando que quede entre 0 y 12 (si $f\ n < 0$ debe devolver 0, si $f\ n > 12$ debe devolver 12). P.ej.

```
Main> flimitada (*2) 9
12
pues  $9*2 = 18 > 12$ 
```

```
Main> flimitada (+(-4)) 3
0
pues  $3-4 = -1 < 0$ 
```

```
Main> flimitada (*2) 5
10
pues  $5*2 = 10$  que está en rango Hacerlo en una sola línea y sin guardas.
Ayuda: usar min y max.
```

- a. Definir una función **cambiarHabilidad/2**, que reciba una función f y una lista de habilidades, y devuelva el resultado de aplicar f con las garantías de rango que da **flimitada**. P.ej.

```
Main> cambiarHabilidad (*2) [2,4,6,8,10]
[4,8,12,12,12]
```

- b. Usar **cambiarHabilidad/2** para llevar a 4 a los que tenían menos de 4, dejando como estaban al resto. P.ej.

```
Main> cambiarHabilidad ... [2,4,5,3,8]
[4,4,5,4,8]
```

Lo que hay que escribir es completar donde están los puntitos.

14. Investigar lo que hace la función **takeWhile/2**, que está incluida en el prelude. Preguntar primero el tipo, y después hacer pruebas. Ayudarse con el nombre.

15. Usar **takeWhile/2** para definir las siguientes funciones: **primerosPares/1**, que recibe una lista de números y devuelve la sublista hasta el primer no par exclusive. P.ej.

```
Main> primerosPares [4,12,3,8,2,9,6]
devuelve [4,16], corta en 3 porque no es par
```

primerosDivisores/2, que recibe una lista de números y un número n, y devuelve la sublista hasta el primer número que no es divisor de n exclusive. P.ej.

```
Main> primerosDivisores 60 [4,12,3,8,2,9,6]
devuelve [4,12,3], corta en 8 porque no divide a 60
```

primerosNoDivisores/2, que recibe una lista de números y un número n, y devuelve la sublista hasta el primer número que sí es divisor de n exclusive. P.ej.

```
Main> primerosNoDivisores 60 [8,9,4,12,3,8,2,9,6]
devuelve [8,9], corta en 4 porque divide a 60
```

16. Se representa la información sobre ingresos y egresos de una persona en cada mes de un año mediante dos listas, de 12 elementos cada una. P.ej., si entre enero y junio gané 100, y entre julio y diciembre gané 120, mi lista de ingresos es:

```
[100,100,100,100,100,100,120,120,120,120,120,120]
```

Si empecé en 100 y fui aumentando de a 20 por mes, llegando a 220, queda:

```
[100,120..220]
```

Y si es al revés, empecé en 220 y fui bajando de a 20 por mes hasta llegar a 100, queda:

```
[220,200..100]
```

(jugar un poco con esta notación)

Definir la función: **huboMesMejorDe/3**, que dadas las listas de ingresos y egresos y un número, devuelve True si el resultado de algún mes es mayor que el número. P.ej.

```
Main> huboMesMejorDe [1..12] [12,11..1] 10
True
```


Porque en diciembre el resultado fue $12-1=11 > 10$.

17. En una población, se estudió que el crecimiento anual de la altura de las personas sigue esta fórmula de acuerdo a la edad:

1 año: 22 cm

2 años: 20 cm

3 años: 18 cm

... así bajando de a 2 cm por año hasta

9 años: 6 cm

10 a 15 años: 4 cm

16 y 17 años: 2 cm

18 y 19 años: 1 cm

20 años o más: 0 cm

A partir de esta información:

- a. Definir la función **crecimientoAnual/1**, que recibe como parámetro la edad de la persona, y devuelve cuánto tiene que crecer en un año. Hacerlo con guardas. La fórmula para 1 a 10 años es $24 - (\text{edad} * 2)$.

- b. Definir la función **crecimientoEntreEdades/2**, que recibe como parámetros dos edades y devuelve cuánto tiene que crecer una persona entre esas dos edades. P.ej.

```
Main> crecimientoEntreEdades 8 12
```

```
22
```

es la suma de $8 + 6 + 4 + 4$, crecimientos de los años 8, 9, 10 y 11 respectivamente.

Nota: Definir la función **crecimientoEntreEdades** en una sola línea, usando map y suma.

- c. Armar una función **alturasEnUnAnio/2**, que dada una edad y una lista de alturas de personas, devuelva la altura de esas personas un año después. P.ej.

```
Main> alturasEnUnAnio 7 [120,108,89]
```

```
[130,118,99]
```

Qué es lo que van a medir las tres personas un año después, dado que el coeficiente de crecimiento anual para 7 años da 10 cm.

Nota: definir la función **alturasEnUnAnio** en una sola línea, usando map y la función (+ expresión).

- d. Definir la función **alturaEnEdades/3**, que recibe la altura y la edad de una persona y una lista de edades, y devuelve la lista de la altura que va a tener esa persona en cada una de las edades. P.ej.

```
Main> alturaEnEdades 120 8 [12,15,18]
```

```
[142,154,164]
```

que son las alturas que una persona que mide 120 cm a los 8 años va a tener a los 12, 15 y 18 respectivamente.

18. Se tiene información de las lluvias en un determinado mes por Ej: se conoce la siguiente información:

lluviasEnero = [0,2,5,1,34,2,0,21,0,0,0,5,9,18,4,0]

- a. (muy difícil) Definir la función **rachasLluvia/1**, que devuelve una lista de las listas de los días seguidos que llovió. P.ej. se espera que la consulta

```
Main> rachasLluvia lluviasEnero
```

```
[[2,5,1,34,2],[21],[5,9,18,4]].
```

A partir de esta definir **mayorRachaDeLluvias/1**, que devuelve la cantidad máxima de días seguidos que llovió. P.ej. se espera que la consulta `mayorRachaDeLluvias lluviasEnero` devuelva 5.

Ayuda: ver las funciones `dropWhile` y `takeWhile`, probar p.ej. esto:

```
takeWhile even [2,4,7,10,14,15]
```

```
dropWhile even [2,4,7,10,14,15]
```

19. Definir una función que sume una lista de números.

Nota: Resolverlo utilizando `foldl/foldr`.

20. Definir una función que resuelva la productoria de una lista de números.

Nota: Resolverlo utilizando `foldl/foldr`.

21. Definir la función **dispersion**, que recibe una lista de números y devuelve la dispersión de los valores, o sea máximo - mínimo.

Nota: Probar de utilizar `foldr`.