Auxiliar 2

High Order Functions, Currying, Point-free style

Profesor: Federico Olmedo
Auxiliar: Damián Árquez

P1 - Currying

En clases se vió la función `curryMe` que recibía una función no currificada y retornaba una nueva versión de la función currificada. Ahora, escriba una función `uncurryMe` que, al contrario, reciba una función currificada y retorne su versión no currificada. Note que hay más de una forma de escribir la función, ¿Cuál es la versión más coherente a la semántica de `uncurryMe`?

P2 - Pointfree style

Escriba una función `process :: (a -> b) -> (a -> Bool) -> [a] -> [b]` que reciba una función (`(a -> b)`), un predicado (`a -> Bool`) y una lista (`[a]`) y retorne una nueva lista con todos los elementos que cumplan el predicado (`[b]`), aplicandoles la función. Defina 3 versiones de la función:

1. `processR` de manera recursiva y utilizando *Conditional clauses*.
2. `processMF` utilizando las funciones `map` y `filter`.
3. `processPF` utilizando `map` y `filter` pero con estilo *pointfree*, usando la composición de funciones.

Código 1: sumIntsBetween

```
1 > process (+1) odd [2, 3, 4]
2   [4]
3
```

¿Qué puede decir con respecto a la implementación de `processPF` con respecto a `processR`? ¿Cuál versión es la más compacta? ¿Cuál versión es más flexible? ¿Se puede imaginar alguna desventaja de *pointfree style*?

P3 - High Order Functions

1. Defina la función `sumIntsBetween :: Int -> Int -> Int` que retorna la suma (inclusiva) de todos los enteros entre dos valores `a` y `b`, recursivamente. Asuma que `a` siempre es menor o igual a `b`.

Código 2: sumIntsBetween

```
> sumIntsBetween 0 1
  1
> sumIntsBetween 2 4
  9

```

2. Defina una función `db :: Int -> Int` que retorna el doble de un número.

3. Defina la función `sumDbBetween :: Int -> Int -> Int` que retorna la suma del doble de todos los valores entre dos valores a y b. *Hint*: La implementación debería ser bastante similar a la de `sumIntsBetween`.

4. Defina la función `highOrderSumBetween :: (Int -> Int) -> Int -> Int -> Int` que ademas de tomar dos extremos, `a` y `b`, toma una función (`Int -> Int`) que es aplicada a cada número dentro del rango.

5. Defina la función `hoSumDbBetween :: Int -> Int -> Int` análoga a `sumDbBetween` pero implementada usando `highOrderSumBetween` y `db`.

6. Defina la función `hoSumIntsBetween :: Int -> Int -> Int` análoga a `sumIntsBetween` pero implementada usando `highOrderSumBetween`. Notar que acá no se le aplica ninguna función a los números sumados, intente utilizar una lambda para expresar ese comportamiento.

7. Defina la función `higherOrderSequenceApplication` que debe recibir una función (Int -> Int -> Int) que generalice (parametrice) la suma utilizada en `highOrderSumBetween`, una función (Int -> Int) que se aplice a cada elemento dentro del rango, dos elementos `a` y `b`, y un elemento que se debe retornar en el caso de que $a > b$. Finalmente, el tipo de la función será `(Int -> Int -> Int) -> (Int -> Int) -> Int -> Int -> Int -> Int`.

8. Defina la función `hoFactorial :: Int -> Int` utilizando `highOrderSequenceApplication`.

¿Qué ventaja puede ver en el uso de funciones de alto orden? Mire su implementación de `hoFactorial`, ¿Cuál cree que es el *trade-off* por obtener tal nivel de brevedad en el código?