CS-205 Declarative Programming October 24th 2023
Cécilia Pradic, Monika Seisenberger

Lab 4: higher-order functions and list comprehension

Dear all,

- You can do your lab by editing a single file.
- If you want to sign off, please recall to load your files in `ghci` to check for errors before calling a demonstator over. Additionally, it is also a good idea to test the functions you defined in `ghci` on a variety of examples to double-check they have the behaviour you are expecting.
- You can freely use higher-order functions defined in the standard library that we have seen, recursion and list comprehension for all tasks. For your sake we would encourage you to investigate several possibilities for each questions (there are a multitude of good solutions, some being shorter than others).
- This week is the last opportunity to sign off lab 3 with full marks.

Happy hacking!

Task 4.1: (a) Write a function

```
filterOut :: (a -> Bool) -> [a] -> [a]
```

taking as input a function `p` and a list `xs`, and outputs a list containing the same elements as `xs` *except* those that satisfy `p` (so, intuitively, this would be the complement of `filter`). For instance, `filterOut even [1,2,3]` should evaluate to `[1,3]`

(b) Write a function

```
mapTS :: (a -> b) -> [a] -> [(a,b)]
```

that takes as input a function and a list of elements `xs`, and returns the list of pairs whose first component are taken from `xs` and the second is obtained by appling `f` to it. For instance, `mapTS even [1,2,3]` should evaluate to `[(1,False),(2,True),(3,False)]`.

Task 4.2: Write a function

```
allPairs :: [a] -> [b] -> [(a,b)]
```

that takes as input two lists `xs` and `ys` and outputs a list containing all possible pairings of elements of `xs` and `ys`. For instance, we could have that `allPairs [1,2] ['a','b','c']` evaluates to

```
[(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c')]
```

Task 4.3 A number is *perfect* if it is equal to the sum of its strict divisors (so every divisor except the number itself). For insance, 6 is perfect because it is divisible by $1, 2, 3$ and itself, and we have $6 = 1 + 2 + 3$. Write a function

```
perfects :: Int -> [Int]
```

such that `perfects n` evaluates to the list of all perfect number lesser or equal than n. For instance, `perfects 100` should evaluate to `[6,28]`. What is the value of `perfects 1000`?

Task 4.4 Challenge task:

(a) Write a function

```
allSublistsK :: Int -> [a] -> [[a]]
```

that enumerate all of the possible sublists of size k of a given list. For instance, `allSublistsK 2 [1,2,3]` could evaluate to

```
[[2,3],[1,3],[1,2]]
```

(b) Write a function

```
transpose :: a -> [[a]] -> [[a]]
```

such that, for every index `i` and `j`, if `xs !! i !! j` is defined, then the value of `(transpose x xs) !! j !! i` is defined and equal to `xs !! i !! j`. For instance, `transpose 0 [[1,2,3],[],[4,5]]` could evaluate to

```
[[1,0,4],[2,0,5],[3,0,0]]
```

(c) Consider the representation of polynomials by lists of numbers such that the following function implements evaluation at a point:

```
type Poly a = [a]

evaluateHorner :: Num a => Poly a -> a -> a
evaluateHorner [] _ = 0
evaluateHorner (c : cs) x = c + x * evaluateHorner cs x
```

Under this representation write functions

```
addPoly :: Num a => Poly a -> Poly a -> Poly a
multPoly :: Num a => Poly a -> Poly a -> Poly a
```

that performs the addition and multiplication of polynomials. Check on a number of examples that you have for every values of `p`, `q` and `x`

```
evaluateHorner (addPoly p q) x == evaluateHorner p x + evaluateHorner q x
evaluateHorner (multPoly p q) x == evaluateHorner p x * evaluateHorner q x
```

(d) Using the previous question, attempt to define a `Num` instance for `Poly a` by replacing the comments in the following by actual code

```
instance Num a => Num (Poly a) where
  p + q = -- your code goes here
  ...
```

You should look at the documentation of Num and the example file `types.pl` for further information on what you should be doing. Don't fret too much about `abs`/`signum`, it is a place where there is not necessarily an obvious choice to be made as far as I am aware.