COMP 141: Haskell — Part 6

Instructions: In this exercise, we are going to review a bunch of Haskell structures.

(1) Function `zip` has type `[a] -> [b] -> [(a,b)]`. Give two interpretations of this type expression.

(2) Specify whether these types are the same or not.

  (a) `a -> b -> c` and `b -> a -> c`

  (b) `a -> b -> c` and `a -> (b -> c)`

  (c) `(a -> b) -> c` and `a -> b -> c`

  (d) `(a, b) -> c` and `a -> b -> c`

  (e) `(a -> b) -> (c -> d)` and `(a -> b) -> c -> d`

(3) Check the type for each of the following expressions, and specify in English what does each mean, and what the expression does.

  (a) `(True:)`

  (b) `(:[True,False])`

  (c) `(^4)`

  (d) `(4^)`

(4) Define function `addOrSubtract` that receives a boolean as input. If the input boolean is true then it must return addition function. If the input boolean is false, it must return subtraction function. For example `addOrSubtract True` returns addition function, and thus `addOrSubtract True 5 4` must return `9`. On the other hand, `addOrSubtract False` returns subtraction function, and thus `addOrSubtract False 5 4` must return `1`. *Hint*: The type of the function can be `Num a => Bool -> a -> a -> a` or more specifically `Bool -> Int -> Int -> Int`.

(5) Use `map` to define the following functions.

  (a) Function `listlen :: [[a]] -> [Int]` that receives a list of lists and returns a list of the lengths. For example, if the input is `[[1..6], [5,9,2], [10, 7 .. -5]]` the output must be `[6,3,6]`.

  (b) Function `square :: [Int] -> [Int]` that receives a list of numbers and returns the list of squares. For example, if the input is `[1 .. 8]` then the output must be `[1,4,9,16,25,36,49,64]`.

(6) Use `filter` to define the following functions.

  (a) Function `noSpace :: String -> String` that receives a string and removes all the spaces, i.e., `‘ ’`. For example, if the input is `"My name is Rick"`, then the output must be `"MynameisRick"`.

  (b) Function `lenLT3 :: [[a]] -> [Int]` that receives a list of lists and returns a list of lengths if the length is larger than 3. For example, if the input is `[[1..6], [10,7 .. 0], [1,2], [4,8 .. 22]]` then the output must be `[6,4,5]`.

(7) Define function `listFunc :: [(a -> b -> c)] -> [a] -> [b] -> [c]` that receives a list of functions `fs` and two other list of items `xs` and `ys`, then returns a list resulted by applying

- the first function in `fs` to the first item in `xs` and `ys`,
- the second function in `fs` to the second item in `xs` and `ys`,
- the third function in `fs` to the third item in `xs` and `ys`,

- ...

Use pattern match on lists.

Example: `listFunc [(+), (-), (*)] [1,2,3] [4,5,6]` must return `[5,-3,18]`.

(8) Define function `addFunc3 :: (Num b) => (a -> b) -> (a -> b) -> (a -> b) -> a -> b` that receives three functions f, g, and h, and returns some function that adds the return values of these three functions.

Example: `addFunc3 (+3) (*8) (5-) 7` must return `64`, since (7+3) + (7*8) + (5-7) = 64.

(9) Define function `modList :: Int -> [Int] -> [Int]` that receives a number `n` and a list `xs`, and returns the reminder of devision of every element in `xs` to `n`. Use `map` to define this function.

Example: `modList 3 [5, 11, 28, 30]` must return `[2,2,1,0]`.

(10) Define function `trimAlpha :: String -> String` from an earlier lab, this time using `filter` rather than list comprehensions. This function removes all alphabetic characters from the input string.