Programming languages (TC-2006)

In-class activity 04

Date: September 08, 2020

In this activity, you will practice higher-order functions in the Racket language. Please consider that the purpose of this activity is to allow you to practice and identify strengths and weaknesses. Then, implement these functions as requested and avoid using any built-in functions that already do what you are requested to implement. Since this activity focuses on higher-order functions, consider that, in all the functions requested, you must use, at least, one-higher order function such as `map`, `apply`, `filter`, among others. Failing to use at least one higher-order function in each of the requested functions will cancel any points related to such a function.

1 `multiple7` (15%)

In Racket, the function `filter` takes two arguments: a predicate and a list with the elements to be evaluated according to the predicate provided as first argument. The function `filter` will return a sublist containing only the elements that make the predicate true. For example: `(filter even? '(1 2 3 4 5 6))` returns `'(2 4 6)` since only 2, 4 and 6, are even.

By using higher-order functions such as `map`, `apply` or `filter`, write a function in Racket that uses lambda expressions to return a sublist that contains only the elements which are multiples of seven.

2 `countries` (15%)

By using higher-order functions such as `map`, `apply` or `filter`, write a function in Racket that receives a list of lists, in the form `[[Country, Population]]` that returns the names of the countries with a population larger than a given threshold.

3 `group` (15%)

By using higher-order functions such as `map`, `apply` or `filter`, write a function in Racket that receives two lists of equal length and returns a list that contains lists of two elements (the elements that share the same position in the original lists given as arguments). For simplicity, assume that the lists are always of the same length. So, no additional validations on the input are requested.

4 `combine` (15%)

By using higher-order functions such as `map`, `apply` or `filter`, write a function in Racket that receives two lists of equal length and returns a list containing the values of both lists interchanged (starting with the first element of the list provided as first argument). For simplicity, assume that the lists are always of the same length. So, no additional validations on the input are requested.

5 `sumeven` (20%)

By using higher-order functions such as `map`, `apply` or `filter`, write a function in Racket that receives a list that contains only numbers (no additional verifications on the type of the arguments are requested) and returns the sum of all the even numbers in the list provided as argument.

6 `matrixsum` (20%)

By using higher-order functions such as `map`, `apply` or `filter`, implement a function in Racket that calculates the matrix addition. For simplicity, you can assume that the dimensions of the matrices allow the addition to take place. For this problem, use a 'by row' representation inside the lists. For example, the matrix:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Would be represented in a list as `'((1 4 7) (2 5 8) (3 6 9))`.

Deliverables

Prepare a RKT file that contains the functions requested and submit it to Canvas. **Please, do not submit other formats but RKT**.

I promise to apply my knowledge, strive for its development, and not use unauthorized or illegal means to complete this activity, following the Tecnológico de Monterrey Student Code of Honor.