Cairo University
Faculty of Engineering
Computer Department

Expected Outcome:

This lab is expected to help you understand three concepts of functional programming:

- Currying
- Function Composition
- Refactoring and reuse of Composition blocks.

This lab will help you gain hands-on experience on how to implement them using Ramda JS. Refer to the slides and https://ramdajs.com/docs/

Description:

This pipeline will do the following in order:

1. Filter out numbers which are not divisible by certain value X.
2. Multiply each remaining number by a specific value Y.
3. Add a constant value to each number Z.
4. Repeat steps 1-3 but each time, the values for X, Y & Z may vary.
5. Finally, return the sum of the results.

The multiplication and addition functions must be separate. The pipeline is shown in Fig.1. Incorporate Ramda Js, currying, function composition to write Pointfree style code and write efficient code. The components of the pipeline should be reusable as well as the pipeline.

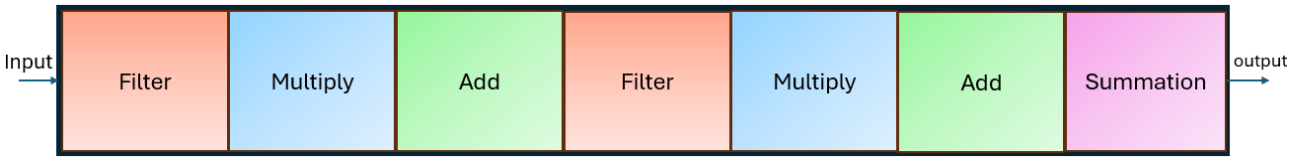

Fig 1 Pipeline chart

Tasks:

1. Implement curried functions named **multiplyBy, isDivisable, and sum**. Each of those functions takes a number **n** as an argument and returns another function. The returned functions will be used with Ramdas' functions to perform the operations required for the pipeline.

2. The list filtering function should be in a separate unit called **filter_divisable** such as it can be possibly used elsewhere and tested separately.

3. Combine the previous pieces to build a pipeline block which will be used in making the final pipeline. The pipeline block should be parameterized blocks for reusability. For example, one can call the pipeline block function to create a block with specified parameters that control the filtering, multiplication and addition operations.

4. Refractor the code using function composition to decrease the overhead of using the map function and combine possible functions together to increase its efficiency without hurting code reusability as much as possible.

5. Implement an aggregation function using **Ramda Reduce** function (**Array_sum**) to sum the elements of the array. This will be the last block in the pipeline. Then build the whole pipeline. There should be a function which creates the whole pipeline such that one can build different pipelines with different parameters.

6. Testing: Ensure your implementation works as expected by writing different test cases for each separate function and block.

7. Use Hindley-Milner for type signature for each function and block created in your code.