Challenge 1: Code analysis

1- What is the advantage of using these kinds of libraries (Ramda for ex) ?

- Ramda makes it simple for us to build complex logic through functional composition .
- By default, all Ramda functions support Currying so that save allot of work .
- Ramda strongly supports Immutable for input parameters and that the some of the core idea of Functional programming paradigm .
- Ramda code return a function and that's a very useful benefit because we can combine it with others to operate on whatever sets of data we choose (reusability) .
- The code is readable and that is a good thing for me as a developer.

2- What is the result of the following code?

```
R.reduce(
  (acc, x) =>
    R.compose(R.flip(R.prepend)(acc), R.sum, R.map(R.add(1)))([x, ...acc]),
  [0]
)([13, 28]);
```

Result : [46, 15, 0]

3- Explain each of the steps the best you can ?

- R.reduce will doing two iteration on [13, 28] list

 # First Iteration : acc =[0] , x = 13 , put values in compose

```
R.compose(R.flip(R.prepend)([0]), R.sum, R.map(R.add(1)))([13, 0]);
```

Note: compose performs right-to-left function composition so we will start debug functions from right to left

1. Add 1 to array items value, **result = [14, 1]** , map on array and add 1 for each element

```
R.map(R.add(1))([13, 0]);
```

2. Adds together all the elements of a list, **result = 15**

```
R.sum()([14, 1]);
```

3. flip will return 15 at the first argument, args will be 15, [0] then prepend will Returns a new list with the given element at the front **result = [15, 0]**

```
R.flip(R.prepend)([0])(15); //result = [15, 0]
```

 # Second Iteration : acc =[15, 0] , x = 28 , put values in compose

```
// Second iteration : acc = [15,0], x = 28
R.compose(R.flip(R.prepend)([15, 0]), R.sum, R.map(R.add(1)))([28, 15, 0]);
```

1. Add 1 to array items value, **result = [29, 16, 1]**, map on array and add 1 for each element

```
R.map(R.add(1))([28, 15, 0]); // result = [29, 16, 1];
```

2. Adds together all the elements of a list, **result = 46**

```
R.sum()([29, 16, 1]); // result = 46
```

3. flip will return 46 at the first argument, args will be 46, [15, 0] then prepend will Returns a new list with the given element at the front **result = [46, 15, 0]**

```
R.flip(R.prepend)([15, 0])(46); // result = [46, 15, 0]
```

Github Repo : https://github.com/Boghdady/typescript-pg-test

References That I Used :

https://ramdajs.com/docs

https://betterprogramming.pub/why-i-fell-in-love-with-ramda-and-functional-programming-in-javascript-797c070133b0

https://fr.umio.us/why-ramda/

https://www.youtube.com/playlist?list=PLrhzvIcii6GMeyUfpn-o5xVCH3_UykrzI