Exercises

Task 1

Implement a higher order function called *sum* in JavaScript that fulfills the following requirement:

```
sum(result => {console.log("-> ", result)}); // -> prints: -> 0
sum(1)(result => {console.log("-> ", result)}); // -> prints: -> 1
sum(1)(2)(result => {console.log("-> ", result)}); // -> prints: -> 3
sum(1)(2)(4)(result => {console.log("-> ", result)}); // -> prints: -> 7
```

Use test driven development here and start with defining a set of Jest test cases for above requirement. Then implement the sum method to have those tests pass.

Task 2

Take the higher order function from Task 1 and make it more flexible. The goal is to allow a user to chain any number of calls together to get the sum of an arbitrary.

```
sum(1)(2)...(4)(result => {console.log("-> ", result)});
// example
sum(1)(2)(3)(4)(5)(6)(result => {console.log("-> ", result)}) // -> prints: -> 21
```

Task 3

Write a function that:

- takes a JavaScript object as first parameter
  - the object can have any structure and depth
- takes a string containing a key as second parameter
- returns a map with all values that are found for that key within the object
  - the key in the resulting map shall be the path to the key within the object where "/" is used as separator
  - the root key for the map is the name of the object

```
/**
 * @param object - object to traverse
 * @param key - keys to search for in object
 * @return map containing all found values
 */
function extractValuesForKey(object, searchKey)
{
    let resultMap = new Map();

    // TODO: your code to parse the object and fill the map

    return resultMap;
}

// Example 1

const result = extractValuesForKey({
  uuid: 1,
  innerOne: {
    someKey: "some text"
  },
  innerTwo: {
    uuid: 2,
    innerThree: {
      someOtherKey: "some other text",
      innerFour: {
        uuid: 3
      }
    }
  }
}, "uuid");

// Example 1 result is a map with the following key value pairs
// "" -> 1
// "innerTwo" -> 2
// "innerTwo/innerThree/innerFour" -> 3


// Example 2

const someObject = {
  uuid: 1,
  innerOne: {
    someKey: "some text"
  },
  innerTwo: {
    uuid: 2,
    innerThree: {
      someOtherKey: "some other text",
      innerFour: {
        uuid: 3
      }
    }
  }
}

const result2 = extractValuesForKey(????, "uuid");

// How do you have to pass someObject to the extractValuesForKey in order to get the following result
// "someObject" -> 1
// "someObject/innerTwo" -> 2
// "someObject/innerTwo/innerThree/innerFour" -> 3
```

Please use test driven development again and also feel free to define additional tests for your code. Find some corner cases, which you might want to test in addition to the above examples.