Higher-order Functions

Functions that operate on other functions, either by taking them as arguments or by returning them, are called higher-order functions.

One area where higher-order functions shine is data processing. This is where functions such as filter, map and reduce can come into play.

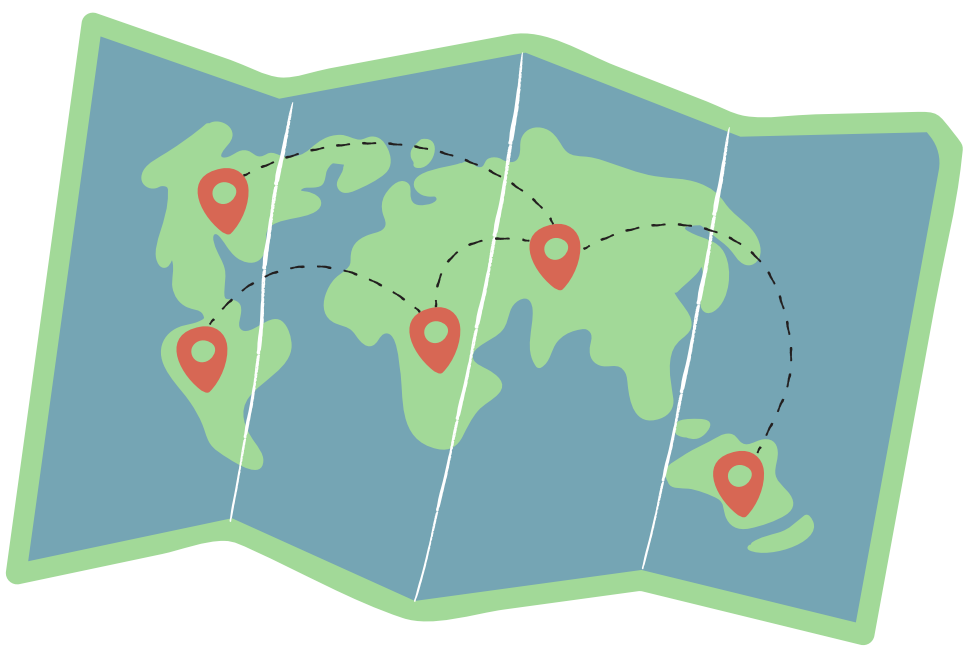
Filter Method

The filter function takes the elements that pass a test created in the function itself, and returns an array comprised of those elements. The function is pure, as in it does not modify the given array.

```
// From the array filter out the pets
    let animals = [{ animal: "lion", isPet: false}, { animal: "cat",
            isPet: true }, { animal: "cheetah", isPet: false }]

// should return the cat animal object
      let housePets = animals.filter((object, index, array) =>
                object.isPet === true);
```

Map Method



The map method transforms an array by applying a function to all of the elements of the array. It returns a new array with the values of the mutated elements from the original array. The new array has the same length as the original one, but the content is different, having been formed by the function within the map method.

```
// Create a new array with values multiplied by 3 from the
numbers array
                    var numbers = [1,2,3,4,5,6,7]

// should return [3,6,9,12,15,18,21]
   var total = numbers.map((num, index, array) => num * 3);
```

Reduce Method

The reduce method takes in an array and return a single value. There are a lot of moving parts with reduce. Firstly there is an accumulator, which is a parameter, that the reduce function runs on for each iteration of the array. Secondly, there is the current item in the array and the current index, that are parameters, and the value of the array itself, which is given as another parameter. These parameters, combined with the starting value given at the end of the function, create the reduce method which can simplify an array to a single value.

```
//Find the total sum of an array using the reduce method.
let array = [1,2,3,4,5];

let answer = array.reduce((sum, num) => sum + num, 0);

// should return 15
console.log(answer);
```

In order to get a single value back from a function, we can combine higher-order functions through chain combinations also known as function chaining.

```
// We want to return a sum of the bread and milk
 let list = [{ item : "bread", total : 2}, { item: "eggs", total : 1 }, {
                                item: "milk", total : 2 }];

// should return 4
     let total = list.filter(( item, index, array) => item.total ===
                    2).reduce((sum, item) => sum + item.total, 0);
```

In this example, we are combining the filter and the reduce methods to an array of objects. Sometimes data can be complex to process, which is why reduce can be used on other higher-order functions. The reduce method is also useful when using a callback function.