Serie N°2: Arrays and Functions

[JavaScript]

Exercise 1: make changes to ***isArrayChanged*** function to check whether two arrays have the different values, with order insensitive.

Note that the function returns true if an array ***before*** has at least one of its values repeated different times in the other array after, else return false.

In other words, we can say that two arrays are equal if:

- Both arrays have the same values even with different positions
- Both arrays have the same number occurrences for each of their values in the other array

```
/**
 * return true if arrays `before` and `after` are diffrent, else false
 * @param {[]} bafore
 * @param {[]} after
 * @returns {bool}
 */
function isArrayChangedOrderInsensitiveS1(before, after) {
    if (before === after) return false;
    if (before.length !== after.length) return true;
    for (const element of before) {
        if (
            before.filter((value) => value == element).length
            !=
            after.filter((value) => value == element).length
        ) return true;
    }
    return false;
}
```

```js
/**
 * return true if arrays `before` and `after` are diffrent, else false
 * @param {[]} bafore
 * @param {[]} after
 * @returns {bool}
 */
function isArrayChangedOrderInsensitiveS2(before, after) {
    if (before === after) return false;
    if (before.length !== after.length) return true;
    let set = new Set([...before, ...after]);
    for (const element of set) {
        if (
            before.filter((value) => value == element).length
            !=
            after.filter((value) => value == element).length
        ) return true;
    }
    return false;
}
```

Exercise 2: make changes to ***isMapChanged*** function to check whether two objects are different by testing specific key-value pairs passed as a parameter in an array ***keys.***

```js
/**
 * Compare two json objects `before` and `after` and returns true if all key values are equal, else false
 * @param {Object} before
 * @param {Object} after
 * @param {[]} keys
 * @returns {bool}
 */
function isMapKeysChanged(before, after, keys) {
    for (const key of keys) {
        if (key && isValueChanged(before[key], after[key])) {
            return true;
        }
    }
    return false;
}
```

Exercise 3: write three JavaScript functions, each takes two arrays ***before*** and ***after*** and:

a) Return an array of removed elements from before
b) Return an array of added elements to after

c) Return an array of common elements between both arrays

```
/**
 * return removed elements from `beforeArray` by comparing it with `afterArray`.
 * @param {[Object]} beforeArray
 * @param {[Object]} afterArray
 * @returns {Object}
 */
function getArrayRemovedElements(beforeArray,afterArray) {
    return beforeArray.filter((value) => {
        return !afterArray.includes(value);
    });
}
```

```
/**
 * return added elements from `beforeArray` by comparing it with `afterArray`.
 * @param {[Object]} beforeArray
 * @param {[Object]} afterArray
 * @returns {Object}
 */
function getArrayAddedElements(beforeArray,afterArray) {
    return afterArray.filter((value) => {
        return !beforeArray.includes(value);
    });
}
```

```
/**
 * return unchanged elements from `beforeArray` by comparing it with
`afterArray`.
 * @param {[Object]} beforeArray
 * @param {[Object]} afterArray
 * @returns {Object}
 */
function getArrayUnchangedElements(beforeArray,afterArray) {
    return afterArray.filter((value) => {
        return beforeArray.includes(value);
    });
}
```

Exercise 4: Write a JavaScript function that takes an array of numbers and a callback function, and return a new array filtered by the callback function.

```js
/**
 * @param {[number]} tab
 * @param {Function} compareFn
 * @returns {[number]}
 */
function filterTabCompare(tab,compareFn) {
    return tab.filter((val) => compareFn(val));
    // or return tab.filter(compareFn);
}

filterTabCompare([1,2,3,4,5], (val) => val % 2 == 0);
```

Exercise 5: add two functions to order object that return the total number of meals and total amount of the order.

```js
let order = {
    meals: [
        {
            name: 'CousCous',
            price: 250,
            amount: 1,
        },
        {
            name: 'Kefta',
            price: 350,
            amount: 1,
        },
        {
            name: 'Pizza',
            price: 250,
            amount: 3,
        },
    ],
};
```

```
let order = {
    ///order parameters
    getTotalMeals: function () {
        let sum = 0;
        this.meals.forEach((meal) => {
            sum += meal.amount;
        });
        return sum;
    },
    getTotalAmount: function () {
        let sum = 0;
        this.meals.forEach((meal) => {
            sum += meal.price*meal.amount;
        });
        return sum;
    },
};
```