


# Coding question -5

Interview questions-53



Follow on   
**@Duvvuru Kishore**



# Separate unique and duplicate values of an array

**Input :** [1, 2, 2, 3, 3, 3, 4, 4, 4, 4];

**Output :**

Unique Values: [ 1 ]

Duplicate Values: [ 2, 3, 4 ]

# Approach 1

```
function separateDuplicates(arr) {  
  const uniqueValues = [];  
  const duplicateValues = [];  
  const count = {};  
  
  for (const item of arr) {  
    if (count[item] === undefined) {  
      count[item] = 1;  
      uniqueValues.push(item);  
    } else {  
      if (count[item] === 1) {  
        const index = uniqueValues.indexOf(item);  
        if (index !== -1) {  
          uniqueValues.splice(index, 1);  
        }  
        duplicateValues.push(item);  
      }  
      count[item]++;  
    }  
  }  
  
  return { uniqueValues, duplicateValues };  
}
```

```
const array = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4];  
const { uniqueValues, duplicateValues } = separateDuplicates(array);  
  
console.log("Unique Values:", uniqueValues);  
console.log("Duplicate Values:", duplicateValues);
```

## Explanation

- **Initialize two arrays:** `uniqueValues` to store unique values and `duplicateValues` to store duplicate values.
- **Create an object called `count`** to keep track of the count of each item in the input array. This object will be used to determine whether an item is unique or a duplicate.
- Use a **`for...of` loop** to iterate through the input array `arr`

- **Inside the loop:**
  - Check if the **count for the current item is undefined**. If it is, it means this is the first occurrence of the item.
    - If it's the first occurrence, add the item to the uniqueValues array and set its count in the count object to 1.
- If the count for the **current item is not undefined**, it means the item has been encountered before.

- Check if the count is 1, indicating the second occurrence of the item.
- If it's the **second occurrence, remove the item from the uniqueValues array** (as it's no longer unique) and add it to the duplicateValues array.
- Increment the count for this item in the count object to keep track of future occurrences.
- Finally, return an object containing both the uniqueValues and duplicateValues arrays.

## Approach 2

```
const inputArray = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4];

const uniqueValues = inputArray.filter((item, index, arr) =>
    arr.indexOf(item) === arr.lastIndexOf(item));

const duplicateValues = Array.from(new Set(inputArray.filter(item =>
    inputArray.indexOf(item) !== inputArray.lastIndexOf(item))));

console.log("Unique Values:", uniqueValues);
console.log("Duplicate Values:", duplicateValues);
```

Explanation 

# Unique values code explanation

- **inputArray.filter(...)** - This starts the filtering process on the inputArray.
- **(item, index, arr) => ...** - This is an arrow function used as the condition for filtering. The filter method will iterate through each element of the inputArray, and for each element, it will pass three arguments to this function:
  - **item** - The current element being processed.
  - **index** - The index of the current element in the array.
  - **arr** - The array being processed, which is inputArray in this case.



- **arr.indexOf(item)** - This part of the condition checks for the first occurrence of item in the array arr. If the item is not found, it returns -1.
- **arr.lastIndexOf(item)** - This part of the condition checks for the last occurrence of item in the array arr. If the item is not found, it returns -1 as well.
- **arr.indexOf(item) === arr.lastIndexOf(item)** - The condition checks whether the first and last occurrence of item in the array are at the same index. If they are the same, it means that item occurs only once in the array, and it is considered a unique value.

# Duplicate explanation

- `inputArray.indexOf(item) !== inputArray.lastIndexOf(item)` - The condition checks whether the first and last occurrence of item in inputArray are at different indices. If they are different, it means that item occurs more than once in the array, making it a duplicate value.
- `new Set(...)` - The code wraps the filtered array with Set. A Set is a built-in JavaScript object that stores unique values. When you create a Set, it automatically eliminates duplicate values from the array and stores only unique values.
- `Array.from(...)` - The code converts the Set back into an array. This step is necessary because a Set doesn't have all the methods and features of an array, so you convert it back to an array for practical use.



Follow on   
**@Duvvuru Kishore**

