05-11-2024

JavaScript Questions - Solved

Biruntha Krishnamoorthy

Batch 13-B

QUESTION 1: REVERSE AN ARRAY

```
function reverseArray(arr) {
   return arr.slice().reverse();
}
```

Explanation:

- arr.slice() - Creates a shallow copy of the array to avoid modifying the original array.
- .reverse() - Reverses the elements in the copied array.

Example:

```
const inputArray = [1, 2, 3, 4, 5];
const reversedArray = reverseArray(inputArray);
console.log(reversedArray); // Output: [5, 4, 3, 2, 1]
```

QUESTION 2: FLATTEN AN ARRAY

```
function flattenArray(arr) {
   return arr.flat(Infinity);
}
```

Explanation:

- .flat() method - The .flat() method in JavaScript creates a new array with all sub-array elements concatenated into it.
- Infinity argument - Passing Infinity as an argument ensures that the array is flattened to a single level, regardless of how deeply nested it is.

Example:

```
const inputArray = [1, [2, 3], [4, [5]]];
const flattenedArray = flattenArray(inputArray);
console.log(flattenedArray); // Output: [1, 2, 3, 4, 5]
```

Explanation of Result:

In this example:

The array [1, [2, 3], [4, [5]]] contains nested arrays.

The flat(Infinity) method goes through each level and flattens the entire structure into [1, 2, 3, 4, 5].

This method is useful for applications where you need all elements at the same level, such as combining multiple lists of items for easy processing or search.

QUESTION 3: CHECK FOR DUPLICATES

```
function hasDuplicates(arr) {
    const uniqueElements = new Set(arr);
    return uniqueElements.size !== arr.length;
}
```

Explanation:

- new Set(arr) - A Set is a data structure that only stores unique values. When you pass an array to a Set, it automatically removes any duplicate elements.
- uniqueElements.size - The size property of the Set will represent the number of unique elements in the array.
- Comparison - If the size of the Set is different from the length of the original array, it means there were duplicates in the array. Therefore, we return true if duplicates exist and false otherwise.

Example:

```
console.log(hasDuplicates([1, 2, 3, 4, 5, 1])); // Output: true
console.log(hasDuplicates([1, 2, 3, 4, 5]));    // Output: false
```

Explanation of Results:

Input [1, 2, 3, 4, 5, 1] - The Set will contain {1, 2, 3, 4, 5}, so uniqueElements.size will be 5, which is less than the array length 6, indicating duplicates.

Input [1, 2, 3, 4, 5] - The Set will contain all elements without removing any, so uniqueElements.size will equal the array length, meaning there are no duplicates.

This function is useful in various scenarios, such as validating user input to ensure all items are unique.

QUESTION 4: MERGE TWO OBJECTS

```
function mergeObjects(obj1, obj2) {
    return { ...obj1, ...obj2 };
}
```

Explanation:

- Spread operator { ...obj1, ...obj2 } - The spread operator (...) copies all properties from each object into a new object.
- Merging Logic - When using the spread operator, if both objects have a property with the same key, the value from the second object (obj2) will overwrite the value from the first object (obj1).

Example:

```
const obj1 = { a: 1, b: 2 };

const obj2 = { b: 2, c: 4 };

const mergedObject = mergeObjects(obj1, obj2);

console.log(mergedObject); // Output: { a: 1, b: 2, c: 4 }
```

Explanation of Result:

The merged object { a: 1, b: 2, c: 4 } includes all properties from obj1 and obj2.

Since both obj1 and obj2 have a property b with the same value (2), this does not cause a conflict, and the value remains as 2.

This method is useful for merging configuration objects or combining data from multiple sources where keys may overlap.

QUESTION 5: FIND THE MAXIMUM NUMBER IN AN ARRAY

```
function findMax(arr) {
   return Math.max(...arr);
}
```

Explanation:

- Spread operator ...arr - The spread operator spreads all elements of the array as individual arguments to the Math.max function.
- Math.max() function - Math.max() takes any number of arguments and returns the largest value among them. When combined with the spread operator, it can find the maximum in an array.

Example:

```
const inputArray = [1, 3, 2, 8, 5];

const maxNumber = findMax(inputArray);

console.log(maxNumber); // Output: 8
```

Explanation of Result:

Input [1, 3, 2, 8, 5] - The function spreads the elements as arguments (1, 3, 2, 8, 5), and Math.max identifies 8 as the largest value.

This approach is efficient for finding the maximum in arrays and avoids the need for looping manually. It's particularly useful for finding the largest value in datasets, such as scores or measurements, where the highest value is needed.

QUESTION 6: GROUP ARRAY OF OBJECTS BY PROPERTY

```
function groupBy(arr, property) {
  return arr.reduce((acc, obj) => {
    const key = obj[property];
    if (!acc[key]) {
      acc[key] = [];
    }
    acc[key].push(obj);
    return acc;
  }, {});
}
```

Explanation:

- reduce method - The reduce method iterates over each item in the array, allowing us to build up a result (acc) as we go.
- Dynamic Key Assignment - For each object (obj), we retrieve the value of the specified property (obj[property]). This value (key) is used as a key in the accumulator object (acc).

Grouping Logic:

If the key doesn't exist in acc, we create it with an empty array (acc[key] = []).

We then push the current object (obj) into the array at that key.

Returning the Accumulator - After iterating over all items, reduce returns acc, which contains the objects grouped by the specified property.

Example:

```
const inputArray = [
  { id: 1, category: 'fruit' },
  { id: 2, category: 'vegetable' },
```

```
  { id: 3, category: 'fruit' }
];
const groupedByCategory = groupBy(inputArray, 'category');
console.log(groupedByCategory);
```

~ Output:

```
{
 fruit: [
  { id: 1, category: 'fruit' },
  { id: 3, category: 'fruit' }
 ],
 vegetable: [
  { id: 2, category: 'vegetable' }
 ]
}
```

Explanation of Result:

For each object in inputArray, we grouped it by the value of the category property.

Objects with "fruit" as their category are grouped together under the fruit key, and objects with "vegetable" as their category are grouped under the vegetable key.

This function is useful for organizing data by categories, tags, or any other property—ideal for applications where data needs to be sorted or displayed in sections based on a common attribute.

QUESTION 7: FIND THE INTERSECTION OF TWO ARRAYS

```
function intersectArrays(arr1, arr2) {
  return arr1.filter(element => arr2.includes(element));
}
```

Explanation:

- filter method - The filter method creates a new array containing only elements that pass a given test.
- arr2.includes(element) - For each element in arr1, we check if it exists in arr2 using the includes method.

If the element from arr1 is found in arr2, it's included in the result array.

Result - The function returns a new array containing only the elements that are present in both arr1 and arr2.

Example:

```
const array1 = [1, 2, 3];

const array2 = [2, 3, 4];

const intersection = intersectArrays(array1, array2);

console.log(intersection); // Output: [2, 3]
```

Explanation of Result:

Input [1, 2, 3] and [2, 3, 4] - The function checks each element of array1:

1 is not found in array2, so it's excluded.

2 and 3 are found in array2, so they're included in the result array.

This function is useful in scenarios where you need to find common elements between two lists, such as finding mutual interests between users, shared tags in posts, or overlapping items in inventories.

QUESTION 8: CALCULATE THE SUM OF ARRAY ELEMENTS

```
function sumArray(arr) {
   return arr.reduce((acc, num) => acc + num, 0);
}
```

Explanation:

- reduce method - The reduce method iterates over each element in the array, accumulating a single result.
- It takes a callback function, which receives an accumulator (acc) and the current element (num).
- Accumulation Logic - In each iteration, the current number (num) is added to the accumulator (acc).
- Initial Value 0 - The 0 at the end of reduce is the initial value of acc. Starting with 0 ensures an accurate sum even if the array is empty.

Example:

```
const inputArray = [1, 2, 3, 4, 5];

const totalSum = sumArray(inputArray);
```

```
console.log(totalSum); // Output: 15
```

Explanation of Result:

Input [1, 2, 3, 4, 5] - The function processes each element:

Starting with 0, it adds 1 (accumulator becomes 1), then 2 (accumulator becomes 3), then 3 (accumulator becomes 6), and so on.

Result 15 - By the end of the array, the sum of all numbers is 15.

This function is useful for calculating total values, such as summing prices in a shopping cart, calculating total scores, or adding up numerical data in financial and statistical applications.

QUESTION 9: REMOVE FALSY VALUES FROM AN ARRAY

```
function removeFalsyValues(arr) {
  return arr.filter(Boolean);
}
```

Explanation:

- filter method - The filter method creates a new array with only elements that pass a given test.
- Passing Boolean as a callback - By passing Boolean as the callback function, each element in the array is converted to a Boolean.
- Only truthy values (like numbers, non-empty strings, etc.) pass the test and are included in the result.
- Falsy values (such as 0, false, '', null, undefined, and NaN) are filtered out.

Example:

```
const inputArray = [0, 1, false, 2, '', 3];
const filteredArray = removeFalsyValues(inputArray);
console.log(filteredArray); // Output: [1, 2, 3]
```

Explanation of Result:

Input [0, 1, false, 2, '', 3] - Each element is tested:

0, false, and '' are falsy, so they're excluded.

1, 2, and 3 are truthy, so they're included in the result.

Result [1, 2, 3] - Only truthy values remain in the array.

This function is useful for cleaning data, especially when dealing with user input or datasets that might contain empty or irrelevant values.

QUESTION 10: CALCULATE AVERAGE OF AN ARRAY

```
function calculateAverage(arr) {
    const sum = arr.reduce((acc, num) => acc + num, 0);
    return sum / arr.length;
}
```

Explanation:

reduce method - The reduce method is used to calculate the sum of all numbers in the array.

It takes an accumulator (acc) and the current element (num) and adds each element to the accumulator.

Initial Value 0 - The initial value of acc is 0, ensuring a correct sum even if the array is empty.

Calculating the Average - Once the sum of all elements is obtained, we divide it by the length of the array (arr.length) to get the average.

If the array is empty, this function would return NaN due to division by zero, so additional handling could be added if necessary.

Example:

```
const inputArray = [1, 2, 3, 4, 5];
const average = calculateAverage(inputArray);
console.log(average); // Output: 3
```

Explanation of Result:

Input [1, 2, 3, 4, 5] - The sum of the elements is calculated as follows:

1 + 2 + 3 + 4 + 5 = 15

Average Calculation - Dividing the sum by the number of elements (15 / 5) gives 3.

This function is useful for finding average values in applications such as grade calculations, temperature readings, or any scenario where the mean value of a dataset is needed.