Week-6 JS-Questions

Question 1: Reverse an Array Problem:

Write a function that takes an array and returns a new array with the elements in reverse order.

Input: [1, 2, 3, 4, 5]

Output: [5, 4, 3, 2, 1]

Use Case: This function can be used in a web application where user reviews need to be displayed in reverse chronological order.

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

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

Explanation;

* The slice() method creates a shallow copy of the input array, ensuring the original array is not modified.
* The reverse() method then reverses the order of elements in the copied array.

Use Case;

This function is useful for displaying items like user reviews or messages in reverse order, particularly fo

Question 2: Flatten an Array Problem:

Write a function that takes a nested array and flattens it to a single-level array.

Input: [1, [2, 3], [4, [5]]]

Output: [1, 2, 3, 4, 5]

Use Case: Useful for aggregating user-selected items from multiple categories into a single list for checkout.

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

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

Explanation

* The flat() method with the argument Infinity recursively flattens all nested levels in the array.

Usecase;

This function is particularly useful for e-commerce or similar applications, where in tems from multiple categories need to be flattened into a single list for easier processing, such as for a checkout or summary page.

Question 3: Check for Duplicates Problem:

Write a function that checks if an array contains duplicates.

Input: [1, 2, 3, 4, 5, 1] Output: true

Input: [1, 2, 3, 4, 5] Output: false

Use Case: Can be used to validate user inputs in forms, such as ensuring usernames are unique during registration.

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

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

Explanation;

* A Set only stores unique values, so converting the array to a Set removes any duplicates.
* We then compare the size of the Set with the original array length. If they differ, it means there were duplicates in the original array.

Usecase;

This function is helpful for validation tasks, such as checking for duplicate entries in user registration forms where each username or email needs to be unique.

Question 4: Merge Two Objects Problem:

Write a function that merges two objects into one.

Input: { a: 1, b: 2 }, { b: 2, c: 4 }

Output: { a: 1, b: 2, c: 4 }

Use Case: This can be used in a web application to combine user profile settings from different sources.

```
function mergeObjects(obj1, obj2) {
 return { ...obj1, ...obj2 };
}
// Example usage
const object1= { a: 1, b: 2 };
const object2 = { b: 2, c: 4 };
const mergedObject = mergeObjects(object1, object2);
console.log(mergedObject); // Output: { a: 1, b: 2, c: 4 }
```

Explanation;

* The function uses the spread operator (`...`) to merge the properties of both objects into a new object.
* If there are duplicate keys, the values from `obj2` will overwrite those from `obj1`.

Usecase;

This function is useful for consolidating user settings or preferences that might come from different sources in a web application, creating a single, unified configuration for the user profile.

Question 5: Find the Maximum Number in an Array Problem:

Write a function that finds the maximum number in an array.

Input: [1, 3, 2, 8, 5]

Output: 8

Use Case: This function can help in analytics dashboards to find the highest sales figure or user activity.

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

// Example usage
const inputArray = [1, 3, 2, 8, 5];
const maxNumber = findMax(inputArray);
console.log(maxNumber); // Output: 8
```

Explanation

* The function uses the Math.max() method combined with the spread operator (...) to find the maximum number in the array.

Usecase;

This function is valuable in analytics, such as finding the peak value in a series of data points, like sales figures or user activity metrics on a dashboard.

Question 6: Group Array of Objects by Property Problem:

Write a function that groups an array of objects by a specific property.

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

Output: {
fruit: [{ id: 1, category: 'fruit' }, { id: 3, category: 'fruit' }],
vegetable: [{ id: 2, category: 'vegetable '}]
}
Use Case: Useful for organizing products by category in an e-commerce application.

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

```
// Example usage
const inputArray = [
    { id: 1, category: 'fruit' },
    { id: 2, category: 'vegetable' },
    { id: 3, category: 'fruit' }
];
const groupedResult = groupByProperty(inputArray, 'category');
console.log(groupedResult);

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

Explanation;

* The function uses reduce() to iterate over the array and build an accumulator object (acc).
   * For each object, it checks if the property value (like fruit or vegetable) exists as a key in the accumulator.
   * If not, it creates an array for that key; then, it pushes the current object into the appropriate array.

Usecase;

This function is especially useful for organizing items, like products in an e-commerce application, by categories (e.g., fruits, vegetables) for easier display and filtering.

Question 7: Find the Intersection of Two Arrays Problem:

Write a function that returns the intersection of two arrays.

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

Output: [2, 3]

Use Case: This can be used in social media applications to find mutual friends between users.

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

// Example usage
const array1 = [1, 2, 3];
const array2 = [2, 3, 4];
const intersection = findIntersection(array1, array2);
console.log(intersection); // Output: [2, 3]
```

Explanation;

* The function uses the filter() method on the first array (arr1) to keep only elements that are also in the second array (arr2), using includes() to check for membership.

Usecase;

This function is ideal for social media applications to find mutual connections, such as identifying mutual friends between two users by comparing their friend lists.

Question 8: Calculate the Sum of Array Elements Problem:

Write a function that calculates the sum of all numbers in an array.
Input: [1, 2, 3, 4, 5]
Output: 15
Use Case: Useful in financial applications to calculate the total expenses or revenue.

```
function calculateSum(arr) {
```

```
Return arr.reduce((acc, num) => acc + num, 0);
}

// Example usage
const inputArray = [1, 2, 3, 4, 5];
const sum = calculateSum(inputArray);
console.log(sum); // Output: 15
```

Explanation;

* The function uses the reduce() method to iterate through the array, adding each element to an accumulator (acc), which starts at 0.

Usecase;

This function is helpful in financial applications for summing up values, such as calculating total expenses or revenue, or aggregating any numeric data.

Question 9: Remove Falsy Values from an Array Problem:

Write a function that removes all falsy values from an array.

Input: [0, 1, false, 2, '', 3]

Output: [1, 2, 3]

Use Case: This function can be used to clean up user inputs or configuration arrays

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

// Example usage
const inputArray = [0, 1, false, 2, '', 3];
const cleanedArray = removeFalsyValues(inputArray);
```

```
console.log(cleanedArray); // Output: [1, 2, 3]
```

Explanation;

* The filter() method is used with Boolean as the callback. In JavaScript, Boolean converts each value to true or false, filtering out all falsy values (0, false, '', null, undefined, NaN).

Usecase;

This function is useful for cleaning arrays of user inputs or configuration settings, ensuring only valid values remain for processing or display.

Question 10: Calculate Average of an Array Problem:

Write a function that calculates the average of all numbers in an array.
Input: [1, 2, 3, 4, 5]
Output: 3
Use Case: This function is useful in educational applications where you need to compute the average score of students from an array of their grades.

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

// Example usage
const inputArray = [1, 2, 3, 4, 5];
const average = calculateAverage(inputArray);
```

```
console.log(average); // Output: 3
```

Explanation;

* The function uses reduce() to calculate the sum of all elements in the array.
   * The sum is then divided by the length of the array to find the average.

Usecase;

This function is particularly useful in educational applications, such as calculating the average score for a student's grades or the average performance metric in data analytics.