**NAME:** MUHAMMAD UZAIR

**NUMBER:** 0323-3358701

**COURSE:** WD-II

**Assignment No.3**

**Github Repository link:**

1. \*\*Map Method:\*\*

- Q: How does the `map` method work in JavaScript, and can you provide an example of when you might use it to manipulate an array of objects?

**Ans:**

const products = [

{ id: 1, name: "Phone", price: 799 },

{ id: 2, name: "Laptop", price: 1299 },

{ id: 3, name: "Headphones", price: 199 },

];

const productNames = products.map(product => product.name);

console.log(productNames);

const formattedPrices = products.map(product => `$${product.price}`);

console.log(formattedPrices);

const productIdsAndNames = products.map(product => ({ id: product.id, name: product.name }));

console.log(productIdsAndNames);

**Output:**

A screen shot of a computer code

Description automatically generated

2. \*\*Filter Method:\*\*

- Q: Explain the purpose of the `filter` method. Provide an example where you use `filter` to extract elements from an array based on a specific condition.

**Ans:**

const numbers = [1, 4, 2, 9, 7, 3];

const evenNumbers = numbers.filter(number => number % 2 === 0);

console.log(evenNumbers);

const products = [

{ id: 1, name: "Phone", price: 799, category: "Electronics" },

{ id: 2, name: "Laptop", price: 1299, category: "Electronics" },

{ id: 3, name: "Book", price: 19, category: "Books" },

];

const electronicsProducts = products.filter(product => product.category === "Electronics");

console.log(electronicsProducts);

**Output:**

A screenshot of a computer screen

Description automatically generated

3. \*\*Sort Method:\*\*

- Q: Discuss the default behavior of the `sort` method for strings and numbers. How would you use a custom comparison function to sort an array of objects by a specific property?

**Ans:**

const products = [

{ name: "Phone", price: 799 },

{ name: "Laptop", price: 1299 },

{ name: "Headphones", price: 199 },

];

products.sort((a, b) => a.name.localeCompare(b.name));

console.log(products);

**Output:**

A screen shot of a computer code

Description automatically generated

4. \*\*Reduce Method:\*\*

- Q: Describe the purpose of the `reduce` method and provide an example where you use it to compute a single value from an array of numbers.

**Ans:**

const numbers = [1, 4, 2, 9, 7];

const sum = numbers.reduce((accumulator, currentValue) => accumulator + currentValue, 0);

console.log(sum);

**Output:**



5. \*\*Find Method:\*\*

- Q: How does the `find` method differ from `filter`? Give an example of a scenario where using `find` is more appropriate than `filter`.

**Ans:**

const users = [

{ id: 1, name: "Alice", isAdmin: false },

{ id: 2, name: "Bob", isAdmin: true },

{ id: 3, name: "Charlie", isAdmin: false },

];

const firstAdmin = users.find(user => user.isAdmin);

console.log(firstAdmin);

const allAdmins = users.filter(user => user.isAdmin);

**Output:**



6. \*\*Combining Methods:\*\*

- Q: Create a chain of array methods (`map`, `filter`, `reduce`, etc.) to transform an array of strings into a single concatenated string with a specific condition.

**Ans:**

const words = ["apple", "banana", "cherry", "orange", "grape", "mango"];

const concatenatedString = words

.filter(word => word.length > 5) // Filter words with more than 5 letters

.map(word => word.toUpperCase()) // Convert words to uppercase

.join("-"); // Join words with hyphens

console.log(concatenatedString);

**Output:**



7. \*\*Callback Functions:\*\*

- Q: Explain the concept of callback functions in the context of array methods. Provide an example of using a callback function with the `map` method.

**Ans:**

const numbers = [1, 2, 3, 4, 5];

const doubledNumbers = numbers.map(number => number \* 2);

console.log(doubledNumbers);

**Output:**



8. \*\*Error Handling:\*\*

- Q: How would you handle potential errors when using array methods like `find` or `reduce`? Provide an example of error handling in such a scenario.

**Ans:**

const users = [];

try {

const firstAdmin = users.find(user => user.isAdmin);

console.log(firstAdmin);

} catch (error) {

console.error("Error: No users found");

}

**Output:**



9. \*\*Immutable Operations:\*\*

- Q: Discuss the importance of immutability when working with array methods. Demonstrate how you would perform immutable operations using methods like `map` or `filter`.

**Ans:**

const products = [

{ name: "Phone", price: 799 },

{ name: "Laptop", price: 1299 },

{ name: "Book", price: 19 },

];

const expensiveProducts = products.filter(product => product.price > 500);

console.log(products);

console.log(expensiveProducts);

**Output:**

A screen shot of a computer code

Description automatically generated

10. \*\*Performance Considerations:\*\*

- Q: Compare the performance implications of using `map` versus `forEach`. In what scenarios would you prefer one over the other, and why?

**Ans:**

Both map and forEach iterate over an array, but their performance and ideal use cases differ:

Performance:

map: Generally slower due to creating a new array with each element. This overhead becomes negligible for small arrays.

forEach: Faster as it doesn't create new arrays, only performing actions on the original array.

Use cases:

Prefer map when:

Creating a new array with transformed elements: Its primary purpose.

Complex transformations: Callback functions have complex logic needing intermediate values.

Chaining methods: Integrates well with filter and reduce.

Prefer forEach when:

Side effects without new arrays: Updating external variables, logging, or DOM manipulation.

Large arrays: No new array creation offers slight performance gains.

Simple iterations: No changes needed, just looping through elements.

In short:

Data transformation or new arrays: Choose map.

Side effects or simple iterations: Choose forEach.

Small arrays: Performance difference negligible.

Choose based on your needs and desired outcome.

1. \*\*Map Transformation:\*\*

- Q: Given an array of integers, use the `map` method to square each element and return a new array with the squared values.

**Ans:**

const numbers = [1, 2, 3, 4, 5];

const squaredNumbers = numbers.map(number => number \* number);

console.log(squaredNumbers);

**Output:**



2. \*\*Filter and Map Combination:\*\*

- Q: Take an array of strings, filter out the ones with a length less than 5, and then capitalize the remaining strings using the `map` method.

**Ans:**

const words = ["hello", "world", "javascript", "programming", "fun"];

const capitalizedLongWords = words

.filter(word => word.length >= 5)

.map(word => word.toUpperCase());

console.log(capitalizedLongWords);

**Output:**



3. \*\*Sorting Objects:\*\*

- Q: Given an array of objects with a 'price' property, use the `sort` method to arrange them in descending order based on their prices.

**Ans:**

const products = [

{ name: "Phone", price: 799 },

{ name: "Laptop", price: 1299 },

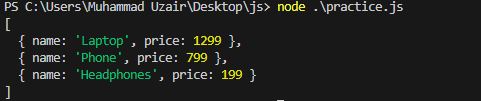
{ name: "Headphones", price: 199 },

];

products.sort((productA, productB) => productB.price - productA.price);

console.log(products);

**Output:**



4. \*\*Reduce for Aggregation:\*\*

- Q: Use the `reduce` method to find the total sum of all even numbers in an array of integers.

**Ans:**

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const sumOfEvenNumbers = numbers.reduce((accumulator, currentNumber) => {

if (currentNumber % 2 === 0) {

accumulator += currentNumber;

}

return accumulator;

}, 0); // Initial value for the accumulator

console.log(sumOfEvenNumbers); // Output: 30

**Output:**

A black and white screen with white text

Description automatically generated

5. \*\*Find and Modify:\*\*

- Q: Given an array of objects with 'id' properties, use the `find` method to locate an object with a specific 'id' and update its 'status' property to 'completed'.

**Ans:**

const tasks = [

{ id: 1, title: "Task 1", status: "pending" },

{ id: 2, title: "Task 2", status: "in progress" },

{ id: 3, title: "Task 3", status: "pending" },

];

const targetId = 2; // ID of the task to update

const taskToUpdate = tasks.find(task => task.id === targetId);

if (taskToUpdate) {

taskToUpdate.status = "completed";

console.log("Task updated:", taskToUpdate);

} else {

console.log("Task not found with ID:", targetId);

}

**Output:**



6. \*\*Chaining Methods:\*\*

- Q: Create a chain of array methods to find the average of all positive numbers in an array of mixed integers and return the result rounded to two decimal places.

**Ans:**

const numbers = [1, -2, 3, 4, -5, 6, 7, 8, 9, 10];

const average = numbers.filter(number => number > 0)

.reduce((sum, number) => sum + number, 0) /

(numbers.filter(number => number > 0).length || 1); // Handle empty array

console.log("Average of positive numbers:", average.toFixed(2));

**Output:**



7. \*\*Conditional Filtering:\*\*

- Q: Implement a function that takes an array of objects with 'age' properties and returns an array of those who are adults (age 18 and above) using the `filter` method.

**Ans:**

function filterAdults(people) {

return people.filter(person => person.age >= 18);

}

const people = [

{ name: "Alice", age: 25 },

{ name: "Bob", age: 16 },

{ name: "Charlie", age: 30 },

];

const adults = filterAdults(people);

console.log("Adults:", adults);

**Output:**



8. \*\*Advanced Sorting:\*\*

- Q: Sort an array of strings based on their lengths in ascending order. If two strings have the same length, maintain their relative order in the sorted array.

**Ans:**

function sortByLength(strings) {

return strings.sort((stringA, stringB) => {

// Prioritize sorting by string length:

const lengthDifference = stringA.length - stringB.length;

if (lengthDifference !== 0) {

return lengthDifference;

} else {

// If lengths are equal, preserve original order:

return strings.indexOf(stringA) - strings.indexOf(stringB);

}

});

}

9. \*\*Nested Array Operations:\*\*

- Q: Given an array of arrays containing numbers, use a combination of array methods to flatten the structure and then calculate the sum of all the numbers.

**Ans:**

const nestedNumbers = [[1, 2, 3], [4, 5], [6, 7, 8, 9]];

const flattenedNumbers = nestedNumbers.flat(); // Flatten the structure

const sum = flattenedNumbers.reduce((accumulator, number) => accumulator + number, 0); // Calculate the sum

console.log("Sum of all numbers:", sum);

**Output:**



10. \*\*Error Handling with Find:\*\*

- Q: Modify the `find` method to handle the scenario where the desired element is not found, returning a custom default object instead.

**Ans:**

function findWithDefault(array, predicate, defaultValue) {

const foundElement = array.find(predicate);

return foundElement || defaultValue;

}

const foundPerson = findWithDefault(people, person => person.name === targetName, defaultPerson);

console.log(foundPerson);

**Output:**

Output: { name: "Not Found", age: 0 }