QUESTIONS

LEVEL EASY

Demo 01

1.Create a simple application that performs various operations on an array of numbers, including finding the maximum, minimum, average, and filtering even numbers.

Requirements:

1. Create a function `arrayOperations(arr)` that:
    a. Takes an array of numbers as input.
    b. Returns an object with the following properties:
        i. `max`: the maximum number in the array.
        ii. `min`: the minimum number in the array.
        iii. `average`: the average of the numbers.
        iv. `evenNumbers`: an array of even numbers from the original array.
2. Test the function with an example array.

Steps to Solve the Exercise

1. **Define the `arrayOperations` Function:**
    a. Use array methods like `Math.max`, `Math.min`, `reduce`, and `filter`.
2. **Test the Functionality:**
    a. Call the function with a sample array and log the results.

Solution

Here's how you can implement this in JavaScript:

```
javascript
// Step 1: Define the arrayOperations Function
function arrayOperations(arr) {
    const max = Math.max(...arr);
    const min = Math.min(...arr);
    const average = arr.reduce((sum, num) => sum + num, 0) /
arr.length;
```

```
    const evenNumbers = arr.filter(num => num % 2 === 0);

    return {
        max,
        min,
        average,
        evenNumbers
    };
}

// Step 2: Test the Functionality
const numbers = [12, 5, 8, 130, 44, 2, 99];
const result = arrayOperations(numbers);

console.log("Array Operations Results:");
console.log(`Max: ${result.max}`);
console.log(`Min: ${result.min}`);
console.log(`Average: ${result.average}`);
console.log(`Even Numbers: ${result.evenNumbers}`);
```

Explanation of the Code

1. **arrayOperations Function:**
    a. This function computes the maximum, minimum, and average of the input array.
    b. It uses the spread operator to find max and min, `reduce` to calculate the average, and `filter` to create an array of even numbers.
2. **Testing the Functionality:**
    a. An example array is defined, and the results of `arrayOperations` are logged to the console.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `array-manipulation`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/array-manipulation.git
cd array-manipulation
```

3. **Create the Files:**

a. Create the necessary files:

```
touch index.js
touch README.md
```

4. **Add Code to index.js:**
    a. Open `index.js` in your preferred code editor and copy the solution code into it.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to use it.

Example content for `README.md`:

```
# Array Manipulation

This project implements simple array operations in JavaScript.

## Features

- Find maximum and minimum values in an array.
- Calculate the average of numbers.
- Filter even numbers from the array.

## How to Use

- Modify the `numbers` array in `index.js` to test with different values.
- Run the application in a JavaScript environment.
```

6. **Stage and Commit Changes:**

```
git add .
git commit -m "Add array manipulation functionality and README"
```

7. **Push to GitHub:**

```
git push origin main
```

8. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.

LEVEL EASY

Demo 02

2.Create a simple program that processes an array of strings to perform various operations such as reversing each string, finding the longest string, and counting the total number of characters across all strings.

Requirements:

1. Create a function `stringArrayOperations(arr)` that:
    a. Takes an array of strings as input.
    b. Returns an object with the following properties:
        i. `reversed`: an array of each string reversed.
        ii. `longest`: the longest string in the array.
        iii. `totalCharacters`: the total number of characters in all strings combined.
2. Test the function with an example array.

Steps to Solve the Exercise

1. **Define the `stringArrayOperations` Function:**
    a. Use array methods like `map`, `reduce`, and a loop to find the longest string.
2. **Test the Functionality:**
    a. Call the function with a sample array and log the results.

Solution

Here's how you can implement this in JavaScript:

```
javascript
// Step 1: Define the stringArrayOperations Function
function stringArrayOperations(arr) {
    const reversed = arr.map(str => str.split('').reverse().join(''));
// Reverse each string
```

```
    const longest = arr.reduce((longestStr, currentStr) =>
        currentStr.length > longestStr.length ? currentStr :
longestStr, '');
    const totalCharacters = arr.reduce((total, str) => total +
str.length, 0);

    return {
        reversed,
        longest,
        totalCharacters
    };
}

// Step 2: Test the Functionality
const fruits = ["apple", "banana", "cherry", "date"];
const result = stringArrayOperations(fruits);

console.log("String Array Operations Results:");
console.log(`Reversed: ${result.reversed}`);
console.log(`Longest String: ${result.longest}`);
console.log(`Total Characters: ${result.totalCharacters}`);
```

Explanation of the Code

1. **stringArrayOperations Function:**
    a. This function performs three main tasks:
        i. Reverses each string in the array using `map`.
        ii. Finds the longest string using `reduce`.
        iii. Calculates the total number of characters in all strings using another `reduce`.
2. **Testing the Functionality:**
    a. An example array of fruits is defined, and the results of `stringArrayOperations` are logged to the console.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `string-array-operations`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/string-array-operations.git
cd string-array-operations
```

3. **Create the Files:**
    a. Create the necessary files:

```
touch index.js
touch README.md
```

4. **Add Code to index.js:**
    a. Open `index.js` in your preferred code editor and copy the solution code into it.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to use it.

Example content for `README.md`:

```
# String Array Operations

This project implements operations on an array of strings in
JavaScript.

## Features

- Reverse each string in the array.
- Find the longest string.
- Count the total number of characters across all strings.

## How to Use

- Modify the `fruits` array in `index.js` to test with different
values.
- Run the application in a JavaScript environment.
```

6. **Stage and Commit Changes:**

```
git add .
git commit -m "Add string array operations functionality and README"
```

7. **Push to GitHub:**

```
git push origin main
```

8. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.

LEVEL EASY

Demo 03

3.Create a small program that uses different types of operators and logs the results to the console.

Requirements:

1. Create a function `operatorDemo()` that:
    a. Demonstrates the use of arithmetic operators (+, -, *, /, %).
    b. Demonstrates the use of assignment operators (=, +=, -=, *=, /=).
    c. Demonstrates the use of comparison operators (==, ===, !=, !==, >, <, >=, <=).
    d. Demonstrates the use of logical operators (&&, ||, !).
2. Log the results of each operation to the console.

Steps to Solve the Exercise

1. **Define the `operatorDemo` Function:**
    a. Implement examples for each type of operator.
2. **Test the Functionality:**
    a. Call the function and observe the logged output.

Solution

Here's how you can implement this in JavaScript:

```
Javascript

// operatorDemo.js
function operatorDemo() {
```

```
    // Arithmetic Operators
    let a = 10;
    let b = 5;

    console.log("Arithmetic Operations:");
    console.log(`Addition: ${a + b}`);          // 15
    console.log(`Subtraction: ${a - b}`);       // 5
    console.log(`Multiplication: ${a * b}`);    // 50
    console.log(`Division: ${a / b}`);          // 2
    console.log(`Modulus: ${a % b}`);           // 0

    // Assignment Operators
    let x = 20;
    console.log("\nAssignment Operations:");
    x += 5;
    console.log(`x += 5: ${x}`);               // 25
    x -= 10;
    console.log(`x -= 10: ${x}`);              // 15
    x *= 2;
    console.log(`x *= 2: ${x}`);                // 30
    x /= 3;
    console.log(`x /= 3: ${x.toFixed(2)}`);     // 10.00

    // Comparison Operators
    console.log("\nComparison Operations:");
    console.log(`10 == '10': ${10 == '10'}`);     // true
    console.log(`10 === '10': ${10 === '10'}`);   // false
    console.log(`10 != '10': ${10 != '10'}`);     // false
    console.log(`10 !== '10': ${10 !== '10'}`);   // true
    console.log(`10 > 5: ${10 > 5}`);               // true
    console.log(`10 < 5: ${10 < 5}`);               // false
    console.log(`10 >= 10: ${10 >= 10}`);           // true
    console.log(`10 <= 5: ${10 <= 5}`);             // false

    // Logical Operators
    console.log("\nLogical Operations:");
    console.log(`true && false: ${true && false}`); // false
    console.log(`true || false: ${true || false}`); // true
    console.log(`!true: ${!true}`);                  // false
}

// Call the function
```

```
operatorDemo();
```

Explanation of the Code

1. **Arithmetic Operations:**
    a. Demonstrates basic arithmetic calculations with two numbers.
2. **Assignment Operations:**
    a. Shows how to use assignment operators to modify a variable.
3. **Comparison Operations:**
    a. Compares values with various comparison operators and logs the results.
4. **Logical Operations:**
    a. Demonstrates logical operations and how they evaluate boolean values.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `operator-demo`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/operator-demo.git
cd operator-demo
```

3. **Create the Files:**
    a. Create the necessary file:

```
touch operatorDemo.js
touch README.md
```

4. **Add Code to operatorDemo.js:**
    a. Open `operatorDemo.js` in your preferred code editor and copy the solution code into it.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to use it.

Example content for `README.md`:

```
# Operator Demo

This project demonstrates the use of various operators in JavaScript.
```

```
## Features

- Arithmetic Operations
- Assignment Operations
- Comparison Operations
- Logical Operations

## How to Use

- Run the script using Node.js:
```

- **Stage and Commit Changes:**

```
git add .
git commit -m "Add operator demonstration functionality and README"
```

- **Push to GitHub:**

```
git push origin main
```

- **View Your Repository:**
  Go back to GitHub and refresh the page to see your new files.

LEVEL MEDIUM

Demo 04

4. Create a simple calculator that performs basic arithmetic operations (addition, subtraction, multiplication, and division) and write test cases for its methods.

Requirements:

1. Create a `Calculator` class with the following methods:
   a. `add(a, b)`: Returns the sum of a and b.

    b. `subtract(a, b)`: Returns the difference of a and b.
    c. `multiply(a, b)`: Returns the product of a and b.
    d. `divide(a, b)`: Returns the quotient of a and b. Handle division by zero appropriately.
2. Write test cases for each method using a testing framework like Jest.

Steps to Solve the Exercise

1. **Define the `Calculator` Class:**
    a. Implement the arithmetic methods.
2. **Set Up Jest for Testing:**
    a. Create test cases for each method in a separate test file.
3. **Test the Functionality:**
    a. Run the tests to verify the correctness of the Calculator methods.

Solution

Step 1: Define the `Calculator` Class

```
javascript
// calculator.js
class Calculator {
    add(a, b) {
        return a + b;
    }

    subtract(a, b) {
        return a - b;
    }

    multiply(a, b) {
        return a * b;
    }

    divide(a, b) {
        if (b === 0) {
            throw new Error("Cannot divide by zero");
        }
        return a / b;
    }
}
```

```
module.exports = Calculator;
```

Step 2: Set Up Jest for Testing

1. Create a test file named `calculator.test.js`.

```
javascript
// calculator.test.js
const Calculator = require('./calculator');

describe('Calculator Tests', () => {
    let calc;

    beforeEach(() => {
        calc = new Calculator();
    });

    test('addition of two numbers', () => {
        expect(calc.add(2, 3)).toBe(5);
    });

    test('subtraction of two numbers', () => {
        expect(calc.subtract(5, 3)).toBe(2);
    });

    test('multiplication of two numbers', () => {
        expect(calc.multiply(4, 5)).toBe(20);
    });

    test('division of two numbers', () => {
        expect(calc.divide(10, 2)).toBe(5);
    });

    test('division by zero throws error', () => {
        expect(() => calc.divide(10, 0)).toThrow("Cannot divide by
zero");
    });
});
```

Explanation of the Code

1. **Calculator Class:**
    a. This class has methods for each arithmetic operation. The `divide` method checks for division by zero and throws an error if attempted.
2. **Test Cases:**
    a. Each method is tested using Jest's `test` function. The `beforeEach` hook initializes a new Calculator instance before each test.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `simple-calculator`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/simple-calculator.git
cd simple-calculator
```

3. **Create the Files:**
    a. Create the necessary files:

```
touch calculator.js
touch calculator.test.js
touch README.md
```

4. **Add Code to calculator.js and calculator.test.js:**
    a. Open `calculator.js` and `calculator.test.js` in your preferred code editor and copy the respective solution code into them.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to run tests.

Example content for `README.md`:

```
# Simple Calculator

This project implements a simple calculator with basic arithmetic
operations in JavaScript.

## Features
```

```
- Addition
- Subtraction
- Multiplication
- Division with error handling for division by zero

## Testing

This project uses Jest for testing. To run tests:

1. Install Jest: `npm install --save-dev jest`
2. Run the tests: `npx jest`
```

6. **Initialize npm and Install Jest:**

```
npm init -y
npm install --save-dev jest
```

7. **Modify package.json:**
    a. Update the `test` script in `package.json`:

```
"scripts": {
    "test": "jest"
}
```

8. **Stage and Commit Changes:**

```
git add .
git commit -m "Add calculator functionality and test cases with
README"
```

9. **Push to GitHub:**

```
git push origin main
```

10. **View Your Repository:**

- Go back to GitHub and refresh the page to see your new files.

LEVEL MEDIUM

Demo 05

5.Create a small program that demonstrates the use of the `push`, `pop`, and `shift` methods on arrays.

Requirements:

1. Create a function `arrayManipulationDemo()` that:
    a. Initializes an array with several string elements.
    b. Uses the `push` method to add new elements to the end of the array.
    c. Uses the `pop` method to remove the last element from the array.
    d. Uses the `shift` method to remove the first element from the array.
    e. Logs the results of each operation to the console.

Steps to Solve the Exercise

1. **Define the `arrayManipulationDemo` Function:**
    a. Implement examples for each array method.
2. **Test the Functionality:**
    a. Call the function and observe the logged output.

Solution

Here's how you can implement this in JavaScript:

```
Javascript
// arrayManipulation.js
function arrayManipulationDemo() {
    // Step 1: Initialize an array
    let fruits = ["apple", "banana", "cherry"];
    console.log("Initial array:", fruits);

   // Step 2: Use push to add elements
    fruits.push("date");
    console.log("After push (add 'date'):", fruits); // ["apple",
"banana", "cherry", "date"]

    // Step 3: Use pop to remove the last element
```

```
    let removedFruit = fruits.pop();
    console.log("After pop (remove last):", fruits); // ["apple",
"banana", "cherry"]
    console.log("Removed fruit:", removedFruit); // "date"

    // Step 4: Use shift to remove the first element
    let firstFruit = fruits.shift();
    console.log("After shift (remove first):", fruits); // ["banana",
"cherry"]
    console.log("Removed first fruit:", firstFruit); // "apple"
}

// Call the function
arrayManipulationDemo();
```

Explanation of the Code

1. **Initialize an Array:**
    a. An initial array of fruits is created.
2. **Using `push`:**
    a. Adds a new fruit to the end of the array.
3. **Using `pop`:**
    a. Removes the last fruit from the array and logs the updated array and the removed item.
4. **Using `shift`:**
    a. Removes the first fruit from the array and logs the updated array and the removed item.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `array-manipulation`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/array-manipulation.git
cd array-manipulation
```

3. **Create the Files:**
    a. Create the necessary file:

```
touch arrayManipulation.js
touch README.md
```

4. **Add Code to arrayManipulation.js:**
    a. Open `arrayManipulation.js` in your preferred code editor and copy the solution code into it.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to run it.

Example content for `README.md`:

```
# Array Manipulation

This project demonstrates the use of array methods in JavaScript: push, pop, and shift.

## Features

- Add elements to an array using push.
- Remove the last element using pop.
- Remove the first element using shift.

## How to Use

- Run the script using Node.js:
```

node arrayManipulation.js

6. **Stage and Commit Changes:**

```
git add .
git commit -m "Add array manipulation functionality and README"
```

7. **Push to GitHub:**

```
git push origin main
```

8. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.

LEVEL MEDIUM

Demo 06

6.Create a small program that demonstrates the use of the `map`, `filter`, and `reduce` methods on arrays.

Requirements:

1. Create a function `arrayMethodsDemo()` that:
    a. Initializes an array of numbers.
    b. Uses the `map` method to create a new array with each number squared.
    c. Uses the `filter` method to create a new array containing only the even numbers.
    d. Uses the `reduce` method to calculate the sum of all numbers in the original array.
2. Log the results of each operation to the console.

Steps to Solve the Exercise

1. **Define the `arrayMethodsDemo` Function:**
    a. Implement examples for each array method.
2. **Test the Functionality:**
    a. Call the function and observe the logged output.

Solution

Here's how you can implement this in JavaScript:

```
javascript
// arrayMethods.js
function arrayMethodsDemo() {
    // Step 1: Initialize an array of numbers
    const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    console.log("Original array:", numbers);

    // Step 2: Use map to create a new array with each number squared
    const squared = numbers.map(num => num * num);
    console.log("Squared numbers:", squared); // [1, 4, 9, 16, 25, 36,
```

```
49, 64, 81, 100]

    // Step 3: Use filter to create a new array containing only even
numbers
    const evenNumbers = numbers.filter(num => num % 2 === 0);
    console.log("Even numbers:", evenNumbers); // [2, 4, 6, 8, 10]

    // Step 4: Use reduce to calculate the sum of all numbers
    const sum = numbers.reduce((total, num) => total + num, 0);
    console.log("Sum of all numbers:", sum); // 55
}

// Call the function
arrayMethodsDemo();
```

Explanation of the Code

1. **Initialize an Array:**
    a. An initial array of numbers from 1 to 10 is created.
2. **Using `map`:**
    a. This method is used to square each number in the array, creating a new array of squared values.
3. **Using `filter`:**
    a. This method filters out the even numbers from the original array, creating a new array that contains only those numbers.
4. **Using `reduce`:**
    a. This method sums all the numbers in the original array, starting from an initial value of 0.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `array-methods-demo`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/array-methods-demo.git
cd array-methods-demo
```

3. **Create the Files:**
    a. Create the necessary file:

```
touch arrayMethods.js
touch README.md
```

4. **Add Code to arrayMethods.js:**
    a. Open `arrayMethods.js` in your preferred code editor and copy the solution code into it.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to run it.

Example content for `README.md`:

```
1. # Array Methods Demo

   This project demonstrates the use of array methods in JavaScript:
   map, filter, and reduce.

   ## Features

   - Square each number using `map`.
   - Filter even numbers using `filter`.
   - Calculate the sum of all numbers using `reduce`.
```

Stage and Commit Changes:

```
git add .
git commit -m "Add array methods demonstration functionality and
README"
```

6. **Push to GitHub:**

```
git push origin main
```

7. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.

LEVEL DIFFICULT

Demo 07

7.Create a program that demonstrates advanced array manipulation techniques in JavaScript.

Requirements:

1. Create a function `advancedArrayDemo()` that:
    a. Initializes a nested array representing a collection of books, where each book is an object with properties: `title`, `author`, and `genres`.
    b. Uses array destructuring to extract information from the nested arrays.
    c. Uses the spread operator to combine multiple arrays of books.
    d. Uses the `map` method to create a new array of book titles.
    e. Uses the `filter` method to find books of a specific genre.
    f. Uses the `reduce` method to count the total number of books.
2. Log the results of each operation to the console.

Steps to Solve the Exercise

1. **Define the `advancedArrayDemo` Function:**
    a. Implement examples for each advanced array manipulation technique.
2. **Test the Functionality:**
    a. Call the function and observe the logged output.

Solution

Here's how you can implement this in JavaScript:

```
Javascript

// advancedArrayManipulation.js
function advancedArrayDemo() {
    // Step 1: Initialize a nested array of books
    const books = [
        { title: "To Kill a Mockingbird", author: "Harper Lee",
```

```
genres: ["Fiction", "Classic"] },
        { title: "1984", author: "George Orwell", genres:
["Dystopian", "Science Fiction"] },
        { title: "The Great Gatsby", author: "F. Scott Fitzgerald",
genres: ["Fiction", "Classic"] },
        { title: "Brave New World", author: "Aldous Huxley", genres:
["Dystopian", "Science Fiction"] },
    ];

    console.log("Original array of books:", books);

    // Step 2: Array destructuring to extract the first book
    const [firstBook] = books;
    console.log("First book title:", firstBook.title); // "To Kill a
Mockingbird"

    // Step 3: Using the spread operator to combine arrays
    const moreBooks = [
        { title: "The Catcher in the Rye", author: "J.D. Salinger",
genres: ["Fiction"] },
        { title: "Fahrenheit 451", author: "Ray Bradbury", genres:
["Dystopian", "Science Fiction"] },];

    const allBooks = [...books, ...moreBooks];
    console.log("Combined array of books:", allBooks);

    // Step 4: Using map to create an array of book titles
    const bookTitles = allBooks.map(book => book.title);
    console.log("Array of book titles:", bookTitles);

    // Step 5: Using filter to find books of a specific genre
    const fictionBooks = allBooks.filter(book =>
book.genres.includes("Fiction"));
    console.log("Fiction books:", fictionBooks);

    // Step 6: Using reduce to count the total number of books
    const totalBooks = allBooks.reduce((count) => count + 1, 0);
    console.log("Total number of books:", totalBooks);
}

// Call the function
advancedArrayDemo();
```

Explanation of the Code

1. **Initialize a Nested Array:**
    a. An array of book objects is created, each with properties for `title`, `author`, and `genres`.
2. **Array Destructuring:**
    a. The first book is extracted from the array using destructuring syntax.
3. **Spread Operator:**
    a. Two arrays of books are combined into one using the spread operator.
4. **Using `map`:**
    a. The `map` method is used to create an array of book titles.
5. **Using `filter`:**
    a. The `filter` method is used to find books that belong to the "Fiction" genre.
6. **Using `reduce`:**
    a. The `reduce` method is used to count the total number of books in the array.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `advanced-array-manipulation`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/advanced-array-manipulation.git
cd advanced-array-manipulation
```

3. **Create the Files:**
    a. Create the necessary file:

```
touch advancedArrayManipulation.js
touch README.md
```

4. **Add Code to advancedArrayManipulation.js:**
    a. Open `advancedArrayManipulation.js` in your preferred code editor and copy the solution code into it.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to run it.

Example content for `README.md`:

```
# Advanced Array Manipulation

This project demonstrates advanced array manipulation techniques in
JavaScript, including destructuring, the spread operator, and higher-
order array methods.

## Features

- Nested arrays representing a collection of books.
- Array destructuring to extract information.
- Combining arrays with the spread operator.
- Creating a list of book titles with `map`.
- Filtering books by genre with `filter`.
- Counting total books using `reduce`.

## How to Use

- Run the script using Node.js:
```

node advancedArrayManipulation.js

6. **Stage and Commit Changes:**

```
git add .
git commit -m "Add advanced array manipulation functionality and
README"
```

7. **Push to GitHub:**

```
git push origin main
```

8. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.

LEVEL DIFFICULT

Demo 08

8.Create a program that demonstrates the use of the `this` keyword in different contexts within JavaScript functions.

Requirements:

1. Create an object `person` with properties `name` and `age`, and a method `greet()` that uses `this` to refer to the object's properties.
2. Create a function `createCounter` that returns an object with a method `increment()` which uses `this` to maintain a count.
3. Demonstrate the behavior of `this` in a regular function vs. an arrow function.

Steps to Solve the Exercise

1. **Define the `person` Object:**
    a. Create an object with a method that uses `this`.
2. **Define the `createCounter` Function:**
    a. Create a function that returns an object to demonstrate `this` in a method.
3. **Test the Functionality:**
    a. Call the methods and observe how `this` behaves.

Solution

Here's how you can implement this in JavaScript:

```
Javascript
// thisKeywordDemo.js
const person = {
    name: "Alice",
    age: 30,
    greet: function() {
        console.log(`Hello, my name is ${this.name} and I am
${this.age} years old.`);
    }
};

function createCounter() {
    let count = 0;
    return {
        increment: function() {
            count++;
            console.log(`Count: ${count}`);
        },
        // Demonstrating arrow function which does not have its own
'this'
        incrementWithArrow: () => {
```

```
            count++;
            console.log(`Count with arrow function: ${count}`);
        }
    };
}

// Testing the `person` object
console.log("Testing the person object:");
person.greet(); // Hello, my name is Alice and I am 30 years old.

// Testing the counter
console.log("\nTesting the counter:");
const counter = createCounter();
counter.increment(); // Count: 1
counter.increment(); // Count: 2

// Testing the arrow function
console.log("\nTesting the arrow function:");
counter.incrementWithArrow(); // Count with arrow function: 3
counter.incrementWithArrow(); // Count with arrow function: 4
```

Explanation of the Code

1. **`person` Object:**
    a. An object `person` is created with properties and a method `greet()`. The method uses `this` to refer to the `name` and `age` properties of the object.
2. **`createCounter` Function:**
    a. This function returns an object with two methods: `increment` (a regular function) and `incrementWithArrow` (an arrow function). The regular function uses `this`, while the arrow function does not have its own `this`, capturing the outer context.
3. **Testing:**
    a. The `greet()` method of the `person` object and the `increment` methods of the returned object from `createCounter` are called to demonstrate the functionality.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `this-keyword-demo`.
2. **Clone the Repository:**

a. Open your terminal and run:

```
git clone https://github.com/yourusername/this-keyword-demo.git
cd this-keyword-demo
```

3. **Create the Files:**
    a. Create the necessary file:

```
touch thisKeywordDemo.js
touch README.md
```

4. **Add Code to thisKeywordDemo.js:**
    a. Open `thisKeywordDemo.js` in your preferred code editor and copy the solution code into it.
5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to run it.

Example content for `README.md`:

```
# This Keyword Demo

This project demonstrates the use of the `this` keyword in JavaScript functions.

## Features

- Understanding `this` in object methods.
- Demonstrating `this` in regular functions and arrow functions.
- Implementing a simple counter with methods.

## How to Use

- Run the script using Node.js:
```

node thisKeywordDemo.js

6. **Stage and Commit Changes:**

```
git add .
git commit -m "Add demonstration of this keyword in functions and
```

README"

7. **Push to GitHub:**

```
git push origin main
```

8. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.

LEVEL DIFFICULT

Demo 09

9. Create a program that demonstrates the use of multidimensional arrays with advanced manipulation techniques.

Requirements:

1. Create a multidimensional array representing a matrix (2D array) of numbers.
2. Write a function `sumMatrix()` that calculates the sum of all the elements in the matrix.
3. Write a function `transposeMatrix()` that returns the transpose of the matrix.
4. Write a function `getRowSum()` that accepts a row index and returns the sum of that specific row.
5. Use destructuring to extract values from the matrix in your functions.

Steps to Solve the Exercise

1. **Define the Multidimensional Array:**
    a. Create a matrix (2D array) with some sample numbers.
2. **Implement Functions:**
    a. Write the `sumMatrix()`, `transposeMatrix()`, and `getRowSum()` functions.
3. **Test the Functionality:**
    a. Call the functions and log the results to the console.

Solution

Here's how you can implement this in JavaScript:

```
Javascript

// multidimensionalArrayDemo.js

// Step 1: Create a multidimensional array (matrix)
const matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

// Step 2: Function to sum all elements in the matrix
function sumMatrix(mat) {
    return mat.reduce((total, row) => {
        return total + row.reduce((rowTotal, num) => rowTotal + num,
0);
    }, 0);
}

// Step 3: Function to transpose the matrix
function transposeMatrix(mat) {
    return mat[0].map((_, colIndex) => mat.map(row => row[colIndex]));
}

// Step 4: Function to get the sum of a specific row
function getRowSum(mat, rowIndex) {
    const [row] = mat[rowIndex]; // Destructuring to get the row
    return row.reduce((total, num) => total + num, 0);
}

// Testing the functions
console.log("Original Matrix:");
console.table(matrix);

const totalSum = sumMatrix(matrix);
console.log("Sum of all elements in the matrix:", totalSum); //
Output: 45

const transposed = transposeMatrix(matrix);
```

```
console.log("Transposed Matrix:");
console.table(transposed);

const rowSum = getRowSum(matrix, 1); // Sum of the second row (index
1)
console.log("Sum of row 1:", rowSum); // Output: 15
```

Explanation of the Code

1. **Define the Matrix:**
    a. A 3x3 matrix is created, which is a two-dimensional array of numbers.
2. **Function `sumMatrix()`:**
    a. This function uses `reduce` twice: first to iterate over the rows and then over the numbers in each row, summing them up.
3. **Function `transposeMatrix()`:**
    a. This function transposes the matrix by mapping over the first row and constructing new rows from each column.
4. **Function `getRowSum()`:**
    a. This function takes a row index, destructures to extract the row, and calculates the sum of its elements.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `multidimensional-array-demo`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/multidimensional-array-
demo.git
cd multidimensional-array-demo
```

3. **Create the Files:**
    a. Create the necessary file:

```
touch multidimensionalArrayDemo.js
touch README.md
```

4. **Add Code to multidimensionalArrayDemo.js:**

a. Open `multidimensionalArrayDemo.js` in your preferred code editor and copy the solution code into it.

5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to run it.

Example content for `README.md`:

```
# Multidimensional Array Demo

This project demonstrates the use of multidimensional arrays in
JavaScript, including advanced manipulation techniques.

## Features

- Initialize a 2D array (matrix).
- Sum all elements in the matrix.
- Transpose the matrix.
- Calculate the sum of a specific row.

## How to Use

- Run the script using Node.js:
```

node multidimensionalArrayDemo.js

6. **Stage and Commit Changes:**

```
git add .
git commit -m "Add multidimensional array functionality and README"
```

7. **Push to GitHub:**

```
git push origin main
```

8. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.

LEVEL DIFFICULT

Demo 10

10.Create a program that simulates a classroom of students, where each student has properties and is part of a multidimensional array representing the classroom.

Requirements:

1. Create a multidimensional array representing a classroom with multiple groups of students.
2. Each student object should contain properties: `name`, `age`, and `grades` (an array of numbers).
3. Write functions to:
    a. Calculate the average grade for each student.
    b. Find the top student in each group based on average grades.
    c. Flatten the classroom data into a single array of student objects.

Steps to Solve the Exercise

1. **Define the Multidimensional Array:**
    a. Create an array of groups, each containing an array of student objects.
2. **Implement Functions:**
    a. Write functions to calculate averages, find the top student, and flatten the data.
3. **Test the Functionality:**
    a. Call the functions and log the results to the console.

Solution

Here's how you can implement this in JavaScript:

```
javascript

// classroomDemo.js

// Step 1: Create a multidimensional array representing the classroom
const classroom = [
    [
```

```
        { name: "Alice", age: 20, grades: [85, 90, 78] },
        { name: "Bob", age: 21, grades: [80, 70, 90] },
    ],
    [
        { name: "Charlie", age: 22, grades: [95, 88, 92] },
        { name: "David", age: 23, grades: [70, 75, 80] },
    ],
];

// Step 2: Function to calculate the average grade for each student
function calculateAverageGrades(classroom) {
    return classroom.map(group => {
        return group.map(student => {
            const total = student.grades.reduce((acc, grade) => acc +
grade, 0);
            const average = total / student.grades.length;
            return { ...student, average }; // Return a new object
with the average
        });
    });
}

// Step 3: Function to find the top student in each group
function findTopStudents(classroom) {
    return classroom.map(group => {
        return group.reduce((topStudent, student) => {
            return student.average > topStudent.average ? student :
topStudent;
        });
    });
}

// Step 4: Function to flatten the classroom data
function flattenClassroom(classroom) {
    return classroom.flat(); // Use flat() to flatten the
multidimensional array
}

// Testing the functions
const studentsWithAverages = calculateAverageGrades(classroom);
console.log("Students with Average Grades:");
console.table(studentsWithAverages);
```

```
const topStudents = findTopStudents(studentsWithAverages);
console.log("Top Students in Each Group:");
console.table(topStudents);

const flattenedClassroom = flattenClassroom(studentsWithAverages);
console.log("Flattened Classroom Data:");
console.table(flattenedClassroom);
```

Explanation of the Code

1. **Define the Classroom:**
    a. A multidimensional array is created, where each sub-array represents a group of students, and each student is an object with properties `name`, `age`, and `grades`.
2. **Function `calculateAverageGrades()`:**
    a. This function calculates the average grade for each student by summing their grades and dividing by the number of grades. It returns a new object including the average.
3. **Function `findTopStudents()`:**
    a. This function iterates through each group and finds the student with the highest average grade using the `reduce` method.
4. **Function `flattenClassroom()`:**
    a. This function uses the `flat()` method to flatten the multidimensional array into a single array of student objects.

Steps to Load the Code to GitHub

1. **Create a New Repository:**
    a. Go to GitHub and create a new repository named `classroom-demo`.
2. **Clone the Repository:**
    a. Open your terminal and run:

```
git clone https://github.com/yourusername/classroom-demo.git
cd classroom-demo
```

3. **Create the Files:**
    a. Create the necessary file:

```
touch classroomDemo.js
touch README.md
```

4. **Add Code to classroomDemo.js:**

a. Open `classroomDemo.js` in your preferred code editor and copy the solution code into it.

5. **Add a README:**
    a. Open `README.md` and write a brief description of the project, including how to run it.

Example content for `README.md`:

```
# Classroom Demo

This project demonstrates the use of multidimensional arrays and
objects in JavaScript to manage student data.

## Features

- Create a multidimensional array of student objects.
- Calculate average grades for each student.
- Find the top student in each group.
- Flatten the classroom data into a single array.

## How to Use

- Run the script using Node.js:
```

node classroomDemo.js

6. **Stage and Commit Changes:**

```
git add .
git commit -m "Add classroom demo functionality and README"
```

7. **Push to GitHub:**

```
git push origin main
```

8. **View Your Repository:**
    a. Go back to GitHub and refresh the page to see your new files.