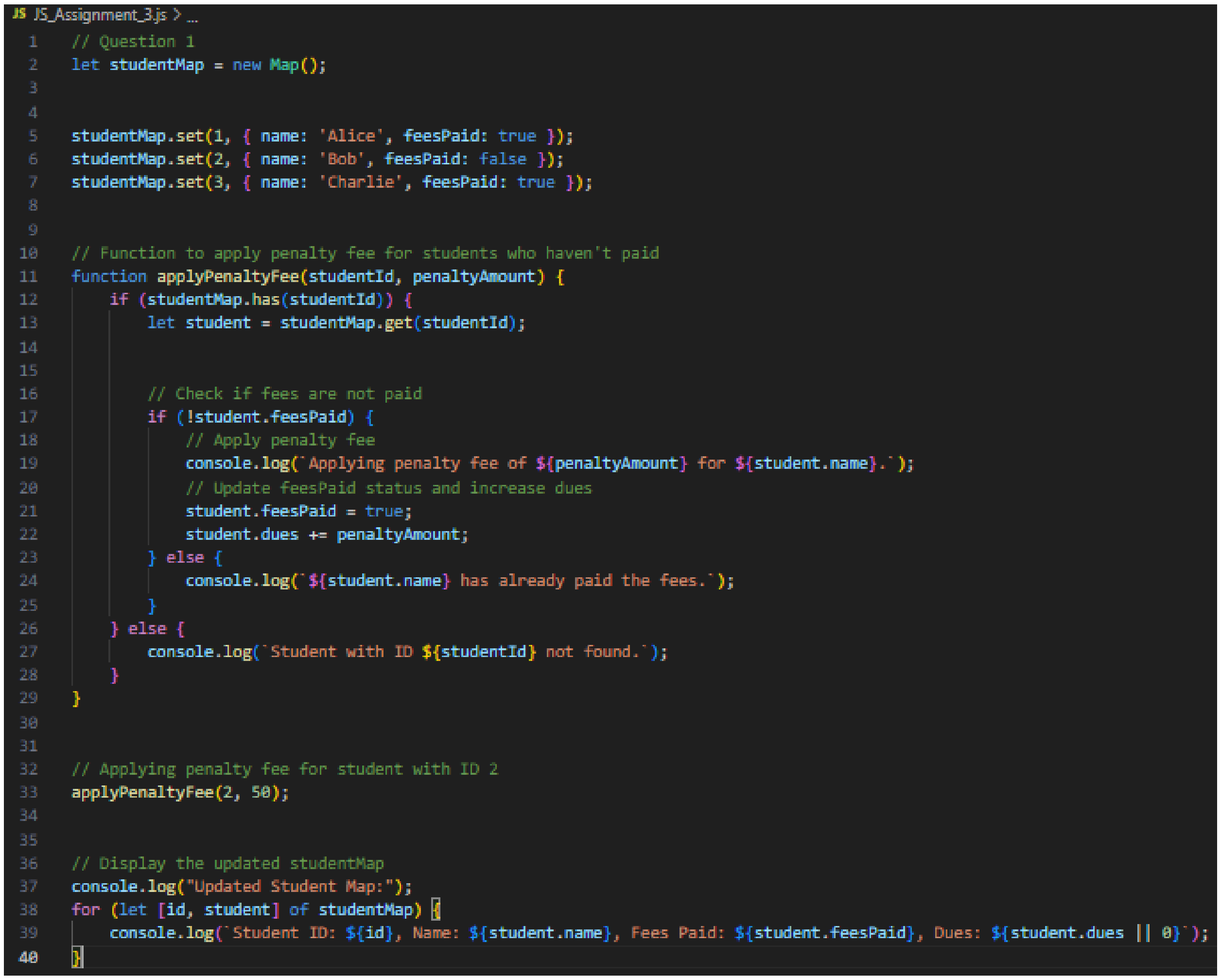
Assignment 3

Q1: 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?

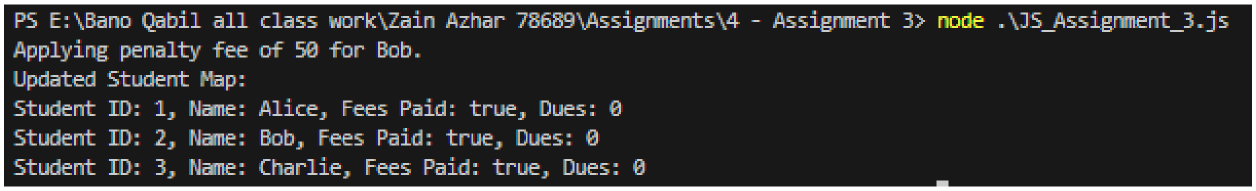
It works by selecting an array, then inside the parameters uses a callback function, which is used to fetch each item inside the selected array and creates a new array meanwhile adding properties in each item such as mathematical operations.

For instance, if we want to add a penalty fee for those students who haven’t paid their fees yet. We can use Map() to locate them and increase their dues.



JS_Assignment_3.js > ...

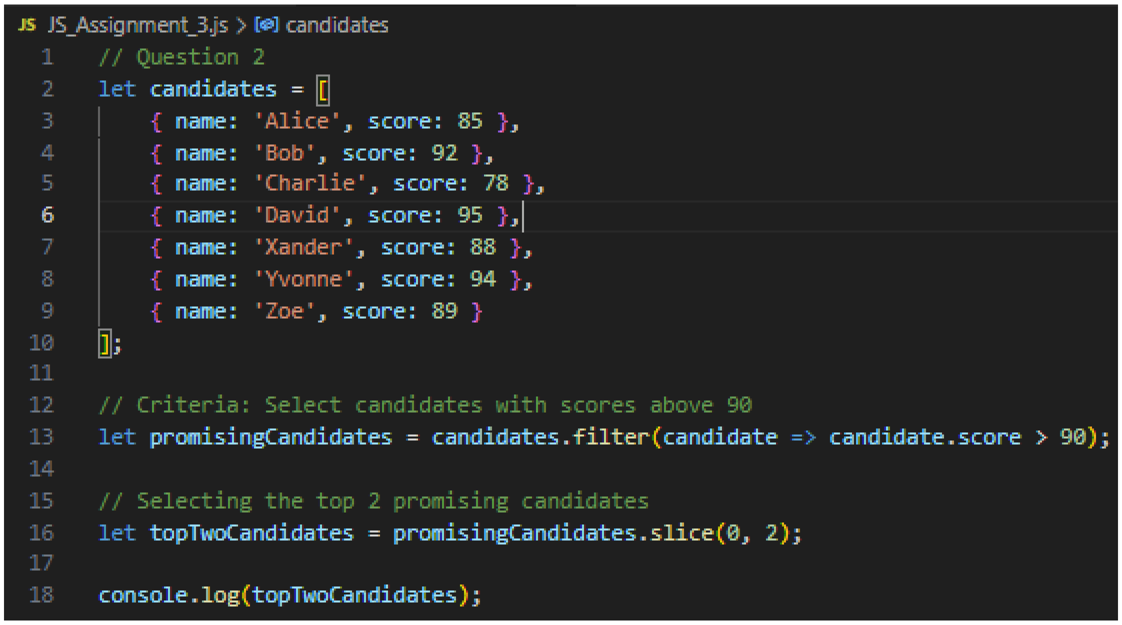
```javascript
// Question 1
let studentMap = new Map();


studentMap.set(1, { name: 'Alice', feesPaid: true });
studentMap.set(2, { name: 'Bob', feesPaid: false });
studentMap.set(3, { name: 'Charlie', feesPaid: true });


// Function to apply penalty fee for students who haven't paid
function applyPenaltyFee(studentId, penaltyAmount) {
    if (studentMap.has(studentId)) {
        let student = studentMap.get(studentId);


        // Check if fees are not paid
        if (!student.feesPaid) {
            // Apply penalty fee
            console.log(`Applying penalty fee of ${penaltyAmount} for ${student.name}.`);
            // Update feesPaid status and increase dues
            student.feesPaid = true;
            student.dues += penaltyAmount;
        } else {
            console.log(`${student.name} has already paid the fees.`);
        }
    } else {
        console.log(`Student with ID ${studentId} not found.`);
    }
}


// Applying penalty fee for student with ID 2
applyPenaltyFee(2, 50);


// Display the updated studentMap
console.log("Updated Student Map:");
for (let [id, student] of studentMap) {
    console.log(`Student ID: ${id}, Name: ${student.name}, Fees Paid: ${student.feesPaid}, Dues: ${student.dues || 0}`);
}
```



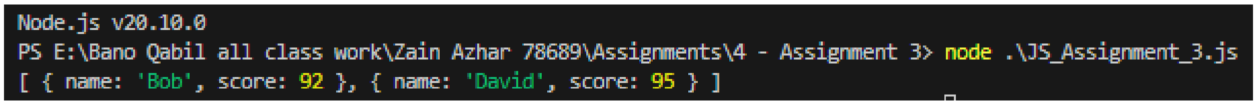
```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
Applying penalty fee of 50 for Bob.
Updated Student Map:
Student ID: 1, Name: Alice, Fees Paid: true, Dues: 0
Student ID: 2, Name: Bob, Fees Paid: true, Dues: 0
Student ID: 3, Name: Charlie, Fees Paid: true, Dues: 0
```

Q2: 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.

It checks each element that matches the condition mentioned in the callback function, then inserts them in a new array and returns it. It is a function which is used in another function as an argument.


JS_Assignment_3.js > candidates

```javascript
// Question 2
let candidates = [
    { name: 'Alice', score: 85 },
    { name: 'Bob', score: 92 },
    { name: 'Charlie', score: 78 },
    { name: 'David', score: 95 },
    { name: 'Xander', score: 88 },
    { name: 'Yvonne', score: 94 },
    { name: 'Zoe', score: 89 }
];

// Criteria: Select candidates with scores above 90
let promisingCandidates = candidates.filter(candidate => candidate.score > 90);

// Selecting the top 2 promising candidates
let topTwoCandidates = promisingCandidates.slice(0, 2);

console.log(topTwoCandidates);
```

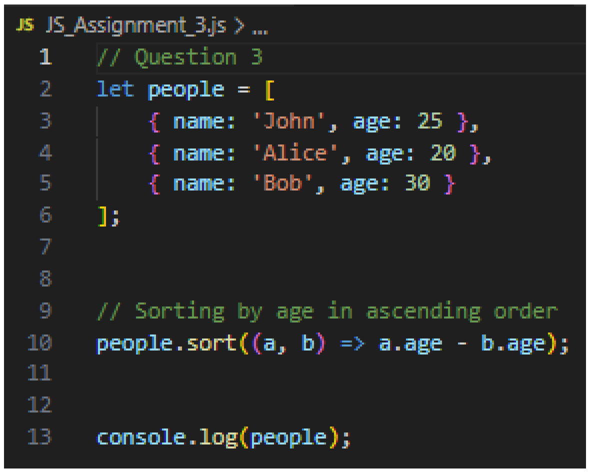


```
Node.js v20.10.0
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[ { name: 'Bob', score: 92 }, { name: 'David', score: 95 } ]
```

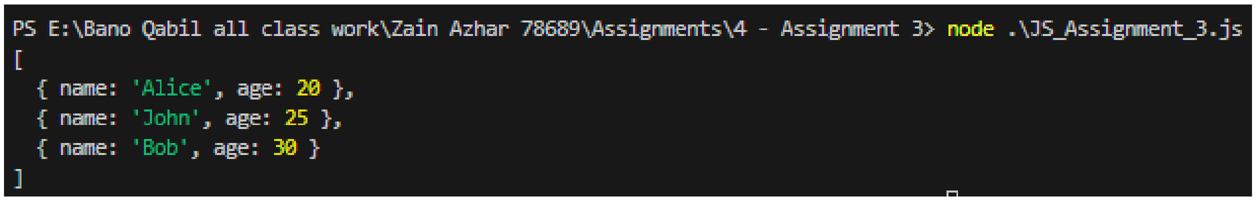
For instance, if we have a list of 7 candidates and are told to find 2 promising ones from further inspection. We can use Filter() to set a criteria and provide an array to find 2 candidates quickly and accurately.

Q3: 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?

The default behavior of Sort() is that it arranges the items present in an array in the ascending order, however an item can't have a string and a number or be the only float among integers.



```
JS_Assignment_3.js > ...
// Question 3
let people = [
    { name: 'John', age: 25 },
    { name: 'Alice', age: 20 },
    { name: 'Bob', age: 30 }
];


// Sorting by age in ascending order
people.sort((a, b) => a.age - b.age);


console.log(people);
```

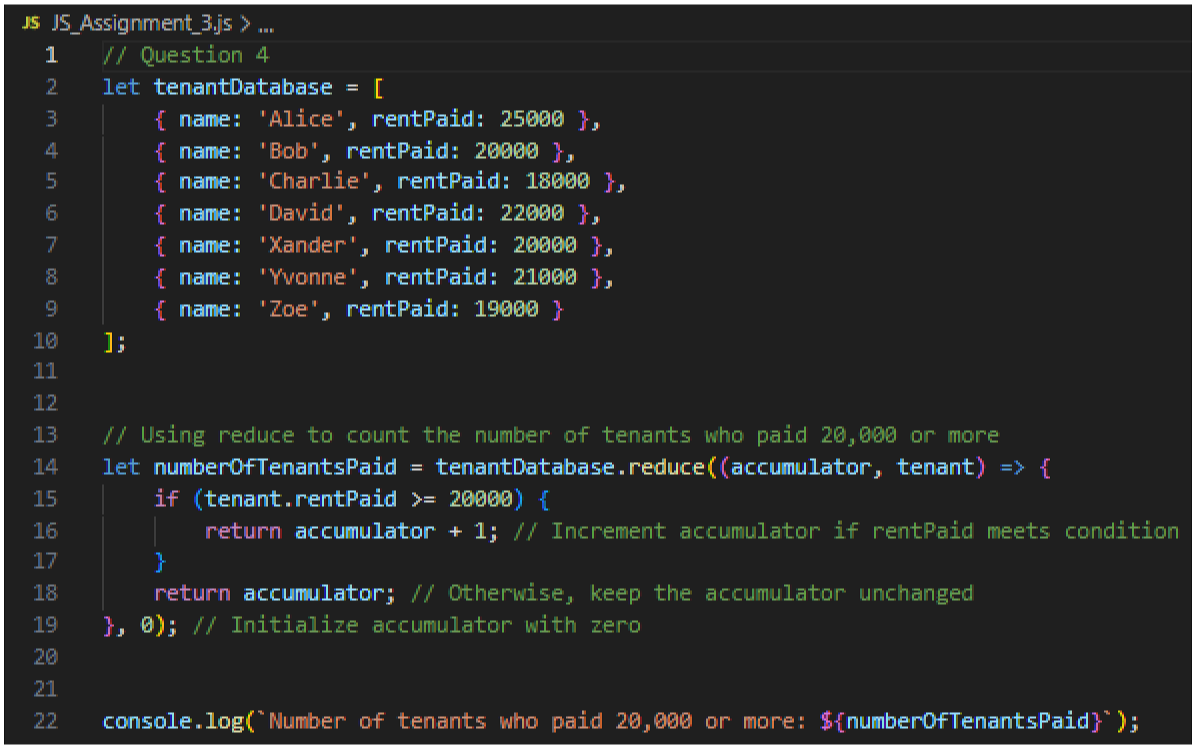


```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[
  { name: 'Alice', age: 20 },
  { name: 'John', age: 25 },
  { name: 'Bob', age: 30 }
]
```

Q4: 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.

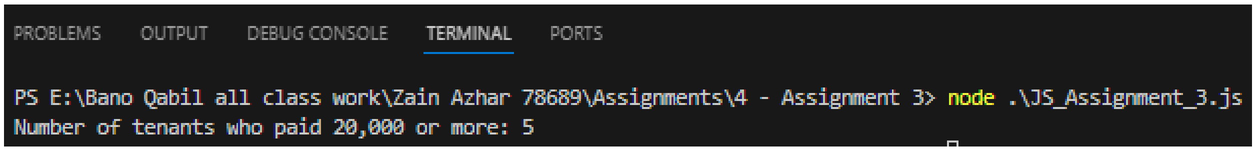
The reduce() method's purpose is to summarize the entire array as a single digit and return it. It takes two parameters, firstly the accumulator, which is the initial value assigned by the programmer, and secondly the current variable, which is the value of each array item.

For instance, if we have a database where we need to count how many people have paid their monthly rent. We can use Reduce() by using an If statement where we initialize the accumulator value to be zero and create the condition e.g. "Has he paid 20,000?". Then each time a condition is true, then the accumulator's value is incremented. This cycle will repeat until the array has ended and the final accumulator's value will determine the exact quantity of paid guests.



JS_Assignment_3.js > ...

```javascript
// Question 4
let tenantDatabase = [
    { name: 'Alice', rentPaid: 25000 },
    { name: 'Bob', rentPaid: 20000 },
    { name: 'Charlie', rentPaid: 18000 },
    { name: 'David', rentPaid: 22000 },
    { name: 'Xander', rentPaid: 20000 },
    { name: 'Yvonne', rentPaid: 21000 },
    { name: 'Zoe', rentPaid: 19000 }
];


// Using reduce to count the number of tenants who paid 20,000 or more
let numberOfTenantsPaid = tenantDatabase.reduce((accumulator, tenant) => {
    if (tenant.rentPaid >= 20000) {
        return accumulator + 1; // Increment accumulator if rentPaid meets condition
    }
    return accumulator; // Otherwise, keep the accumulator unchanged
}, 0); // Initialize accumulator with zero


console.log(`Number of tenants who paid 20,000 or more: ${numberOfTenantsPaid}`);
```

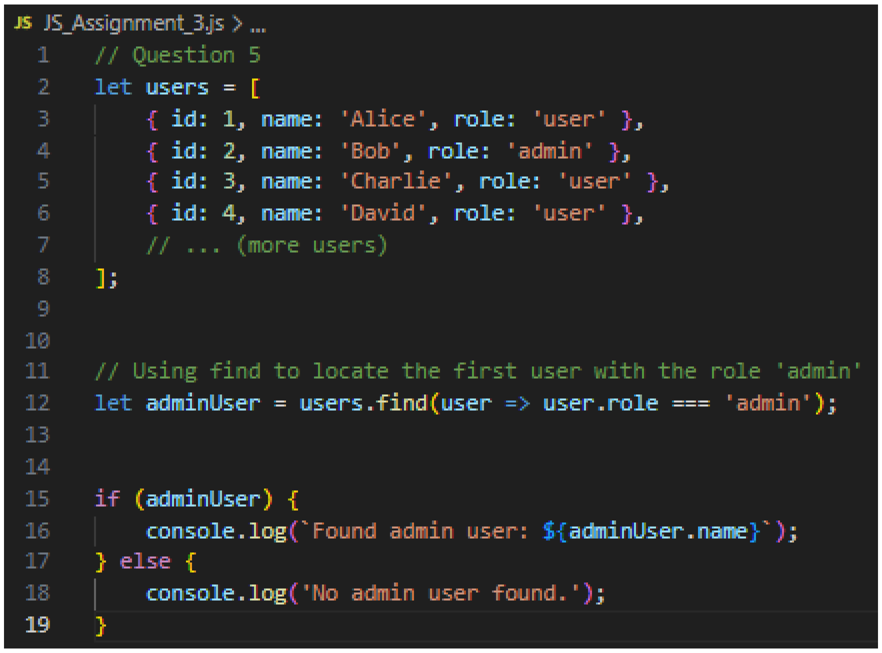


```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
Number of tenants who paid 20,000 or more: 5
```

Q5: How does the `find` method differ from `filter`? Give an example of a scenario where using `find` is more appropriate than `filter`.

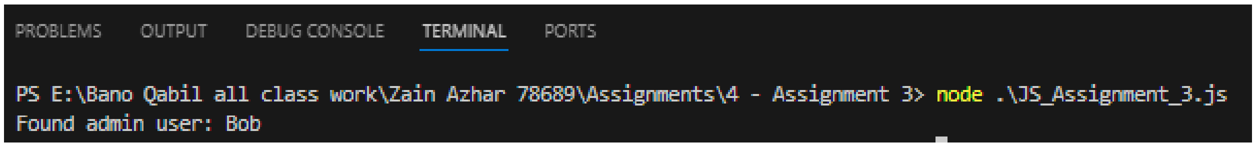
Find() returns only the first variable that matches the condition set in the parameters by the programmer; whereas filter() returns each variable inside that array.

We have an array of users, and we want to find the details of the first user who has a certain role, such as an administrator. Using find in this case is more appropriate because we are interested in a single user who satisfies the condition, not an array of users.



JS_Assignment_3.js > ...

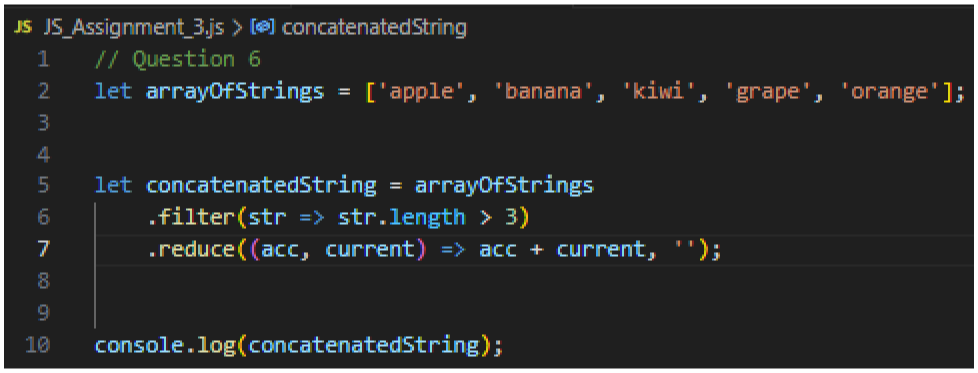
```javascript
// Question 5
let users = [
    { id: 1, name: 'Alice', role: 'user' },
    { id: 2, name: 'Bob', role: 'admin' },
    { id: 3, name: 'Charlie', role: 'user' },
    { id: 4, name: 'David', role: 'user' },
    // ... (more users)
];


// Using find to locate the first user with the role 'admin'
let adminUser = users.find(user => user.role === 'admin');


if (adminUser) {
    console.log(`Found admin user: ${adminUser.name}`);
} else {
    console.log('No admin user found.');
}
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

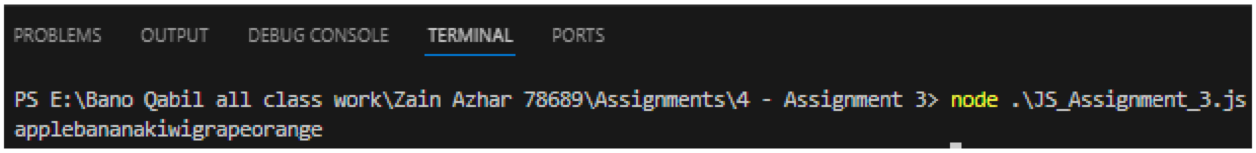
```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
Found admin user: Bob
```

Q6: 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.



JS_Assignment_3.js > concatenatedString

```javascript
// Question 6
let arrayOfStrings = ['apple', 'banana', 'kiwi', 'grape', 'orange'];


let concatenatedString = arrayOfStrings
    .filter(str => str.length > 3)
    .reduce((acc, current) => acc + current, '');



console.log(concatenatedString);
```



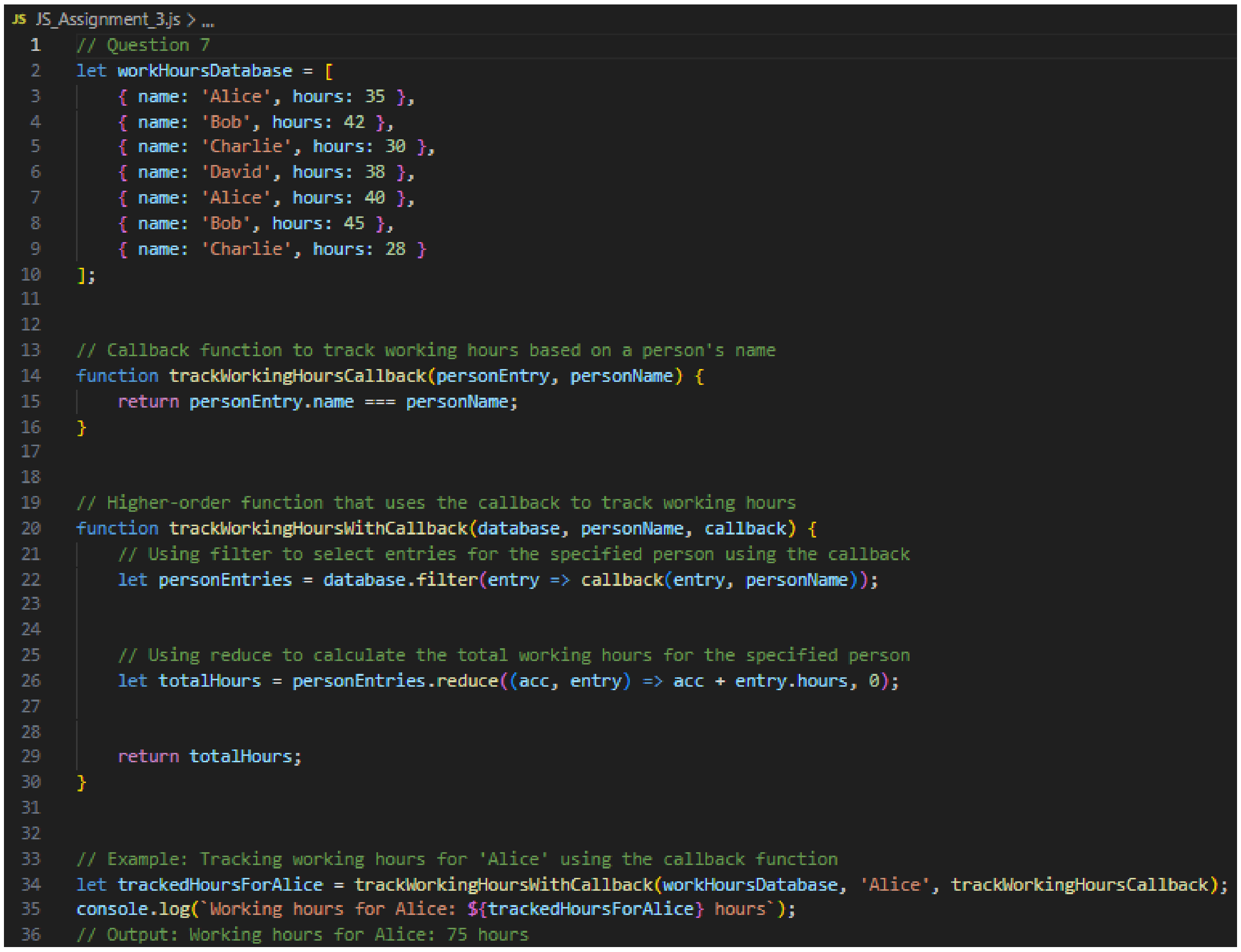
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
applebananakiwigrapeorange
```

Q7: Explain the concept of callback functions in the context of array methods. Provide an example of using a callback function with the `map` method.

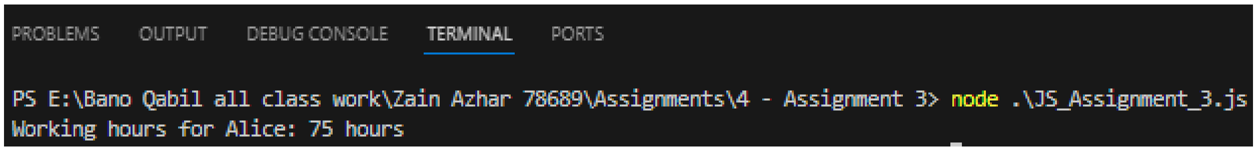
A callback function is a function inside a function which is used as an argument. It is used as an instance for a loop variable, where it fetches each item in an array step by step until it reaches at the end of the array. It is a function inside a function which is used as an argument.

For instance, if we have two columns of a person's name and working hour in a database and want to track the working hours of one person, then we would use that name as a callback.



JS_Assignment_3.js > ...

```javascript
// Question 7
let workHoursDatabase = [
    { name: 'Alice', hours: 35 },
    { name: 'Bob', hours: 42 },
    { name: 'Charlie', hours: 30 },
    { name: 'David', hours: 38 },
    { name: 'Alice', hours: 40 },
    { name: 'Bob', hours: 45 },
    { name: 'Charlie', hours: 28 }
];


// Callback function to track working hours based on a person's name
function trackWorkingHoursCallback(personEntry, personName) {
    return personEntry.name === personName;
}


// Higher-order function that uses the callback to track working hours
function trackWorkingHoursWithCallback(database, personName, callback) {
    // Using filter to select entries for the specified person using the callback
    let personEntries = database.filter(entry => callback(entry, personName));


    // Using reduce to calculate the total working hours for the specified person
    let totalHours = personEntries.reduce((acc, entry) => acc + entry.hours, 0);


    return totalHours;
}


// Example: Tracking working hours for 'Alice' using the callback function
let trackedHoursForAlice = trackWorkingHoursWithCallback(workHoursDatabase, 'Alice', trackWorkingHoursCallback);
console.log(`Working hours for Alice: ${trackedHoursForAlice} hours`);
// Output: Working hours for Alice: 75 hours
```



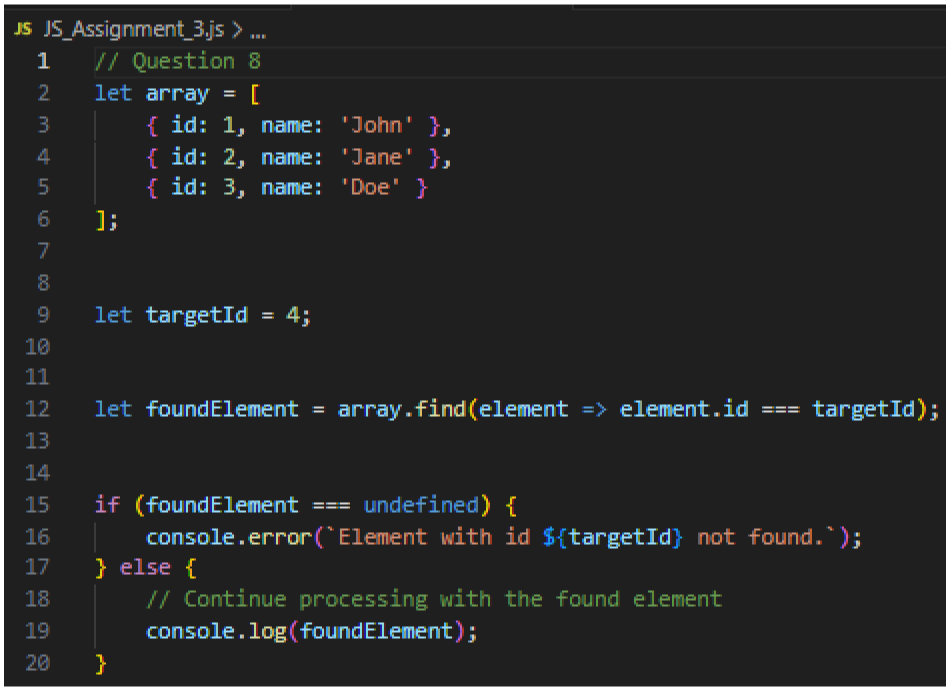
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
Working hours for Alice: 75 hours
```

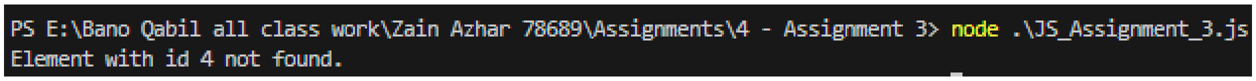
Q8: How would you handle potential errors when using array methods like `find` or `reduce`? Provide an example of error handling in such a scenario.

When using find() we can check if the array is empty or element is not found, and when using reduce() we can check if the array is empty.

For instance:



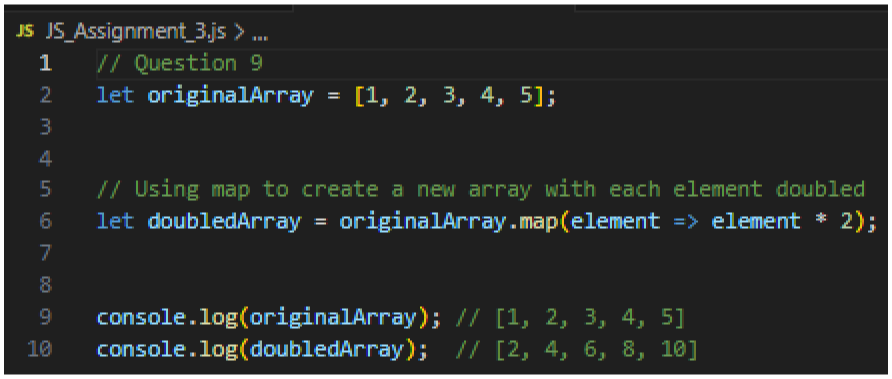
```javascript
// Question 8
let array = [
    { id: 1, name: 'John' },
    { id: 2, name: 'Jane' },
    { id: 3, name: 'Doe' }
];


let targetId = 4;


let foundElement = array.find(element => element.id === targetId);


if (foundElement === undefined) {
    console.error(`Element with id ${targetId} not found.`);
} else {
    // Continue processing with the found element
    console.log(foundElement);
}
```



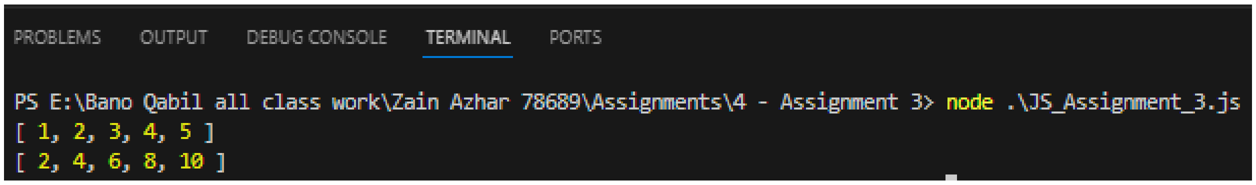
```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
Element with id 4 not found.
```

Q9: Discuss the importance of immutability when working with array methods. Demonstrate how you would perform immutable operations using methods like `map` or `filter`.

It is important because it helps a programmer in debugging because you can check for errors or understand what is the value and where it is stored and in maintainability because it simplifies the maintenance process.



```javascript
// Question 9
let originalArray = [1, 2, 3, 4, 5];


// Using map to create a new array with each element doubled
let doubledArray = originalArray.map(element => element * 2);


console.log(originalArray); // [1, 2, 3, 4, 5]
console.log(doubledArray);  // [2, 4, 6, 8, 10]
```

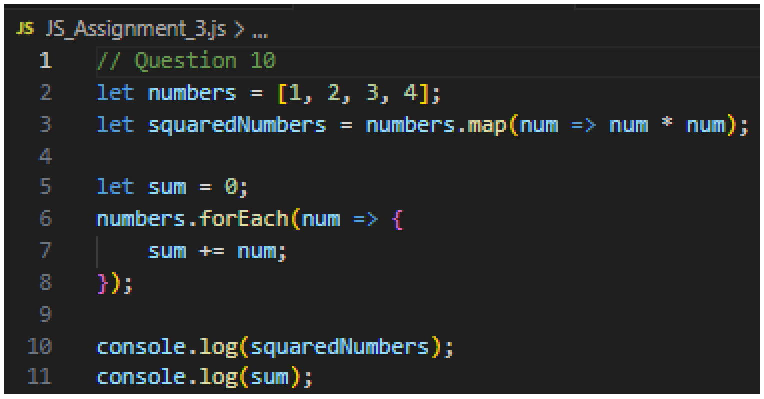


```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[ 1, 2, 3, 4, 5 ]
[ 2, 4, 6, 8, 10 ]
```

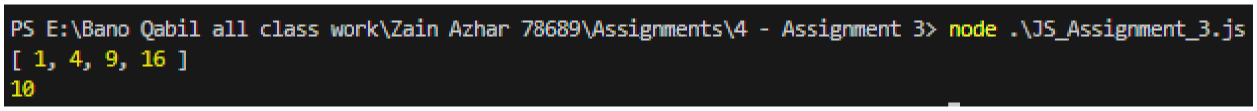
Q10: Compare the performance implications of using `map` versus `forEach`. In what scenarios would you prefer one over the other, and why?

The performance implication of map() is that it creates a new array and returns it, so it's memory-intensive as it allocates memory for the new array. Whereas, forEach() does not create a new array. It simply iterates over the existing array and performs the specified operation for each element.

We can use map() when you want to transform each element in an array and create a new array with the results. Whereas, we can use forEach() when you want to iterate over elements in an array and perform some operation without creating a new array.

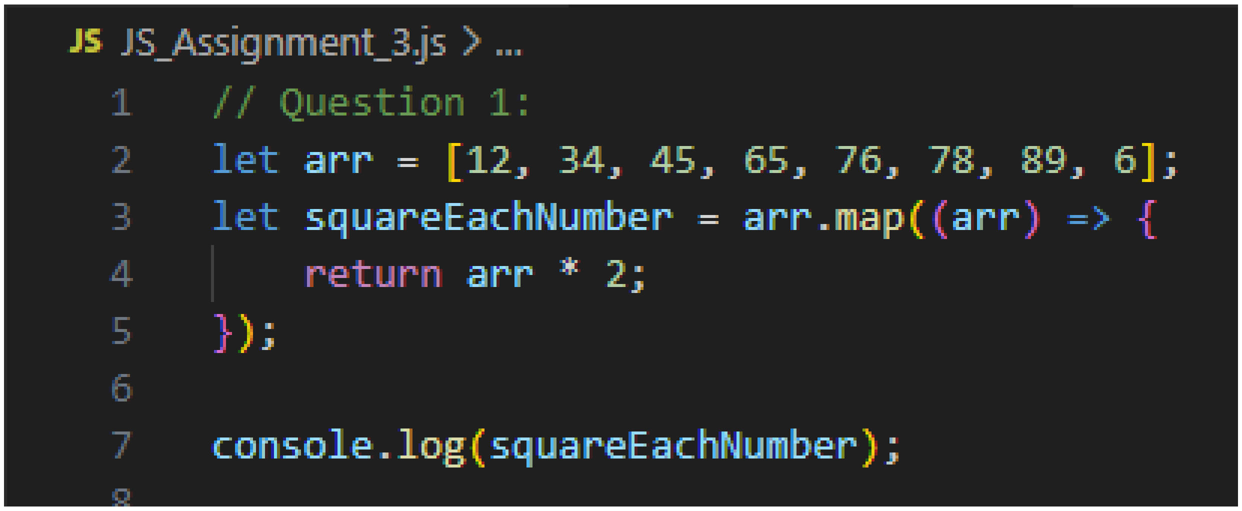


```javascript
// Question 10
let numbers = [1, 2, 3, 4];
let squaredNumbers = numbers.map(num => num * num);

let sum = 0;
numbers.forEach(num => {
    sum += num;
});

console.log(squaredNumbers);
console.log(sum);
```



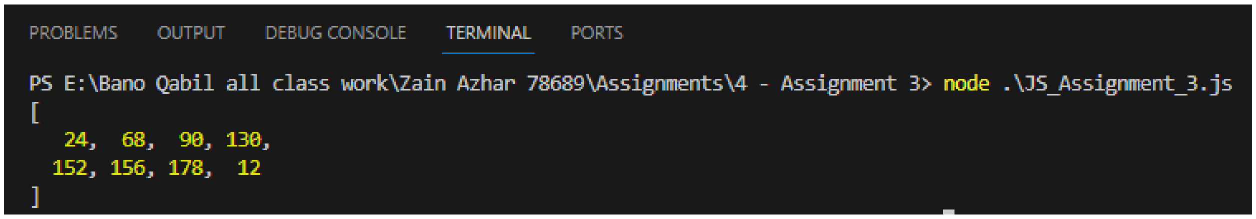
```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[ 1, 4, 9, 16 ]
10
```

Q1: Given an array of integers, use the `map` method to square each element and return a new array with the squared values.



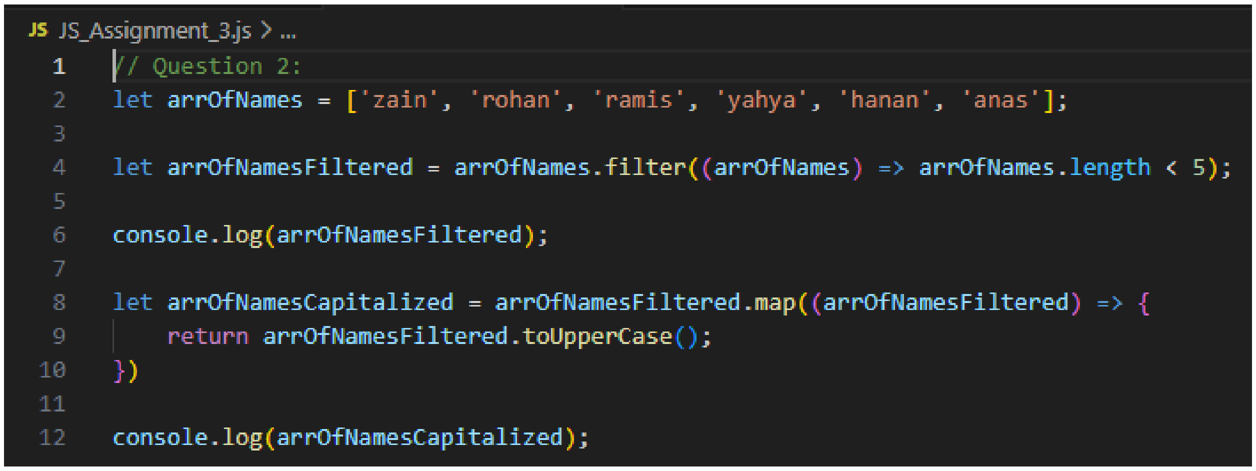
JS_Assignment_3.js > ...

```javascript
// Question 1:
let arr = [12, 34, 45, 65, 76, 78, 89, 6];
let squareEachNumber = arr.map((arr) => {
    return arr * 2;
});

console.log(squareEachNumber);
```



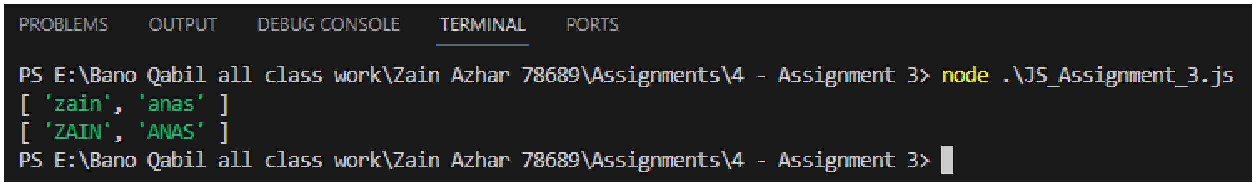
```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[
   24,  68,  90, 130,
  152, 156, 178,  12
]
```

Q2: 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.



JS_Assignment_3.js > ...

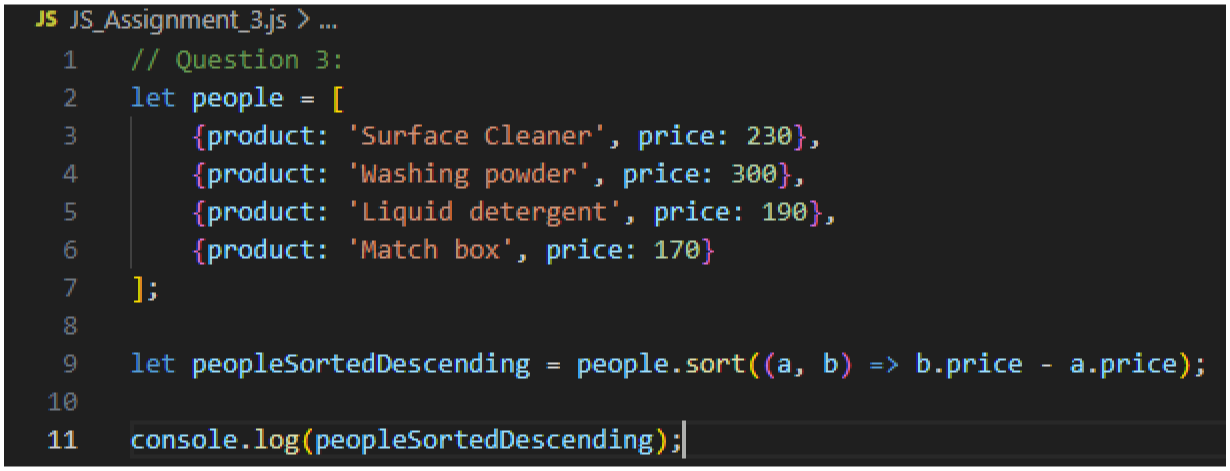
```javascript
// Question 2:
let arrOfNames = ['zain', 'rohan', 'ramis', 'yahya', 'hanan', 'anas'];

let arrOfNamesFiltered = arrOfNames.filter((arrOfNames) => arrOfNames.length < 5);

console.log(arrOfNamesFiltered);

let arrOfNamesCapitalized = arrOfNamesFiltered.map((arrOfNamesFiltered) => {
    return arrOfNamesFiltered.toUpperCase();
})

console.log(arrOfNamesCapitalized);
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

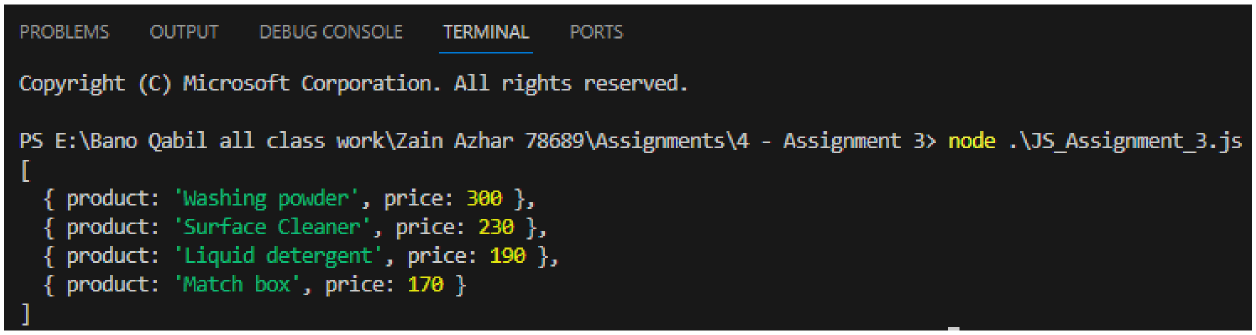
```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[ 'zain', 'anas' ]
[ 'ZAIN', 'ANAS' ]
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3>
```

Q3: Given an array of objects with a 'price' property, use the `sort` method to arrange them in descending order based on their prices.



JS_Assignment_3.js > ...

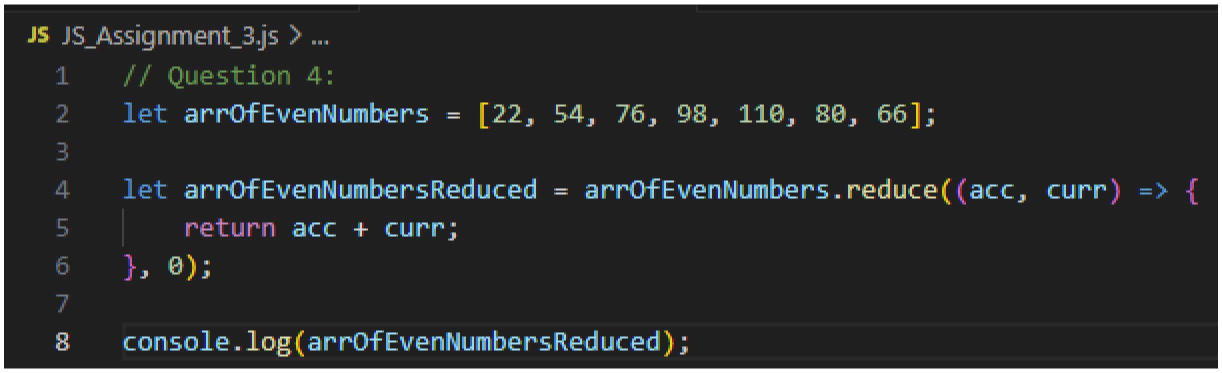
```javascript
// Question 3:
let people = [
    {product: 'Surface Cleaner', price: 230},
    {product: 'Washing powder', price: 300},
    {product: 'Liquid detergent', price: 190},
    {product: 'Match box', price: 170}
];

let peopleSortedDescending = people.sort((a, b) => b.price - a.price);

console.log(peopleSortedDescending);
```



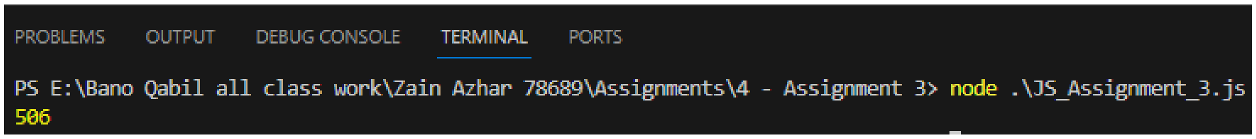
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Copyright (C) Microsoft Corporation. All rights reserved.

PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[
  { product: 'Washing powder', price: 300 },
  { product: 'Surface Cleaner', price: 230 },
  { product: 'Liquid detergent', price: 190 },
  { product: 'Match box', price: 170 }
]
```

Q4: Use the `reduce` method to find the total sum of all even numbers in an array of integers.

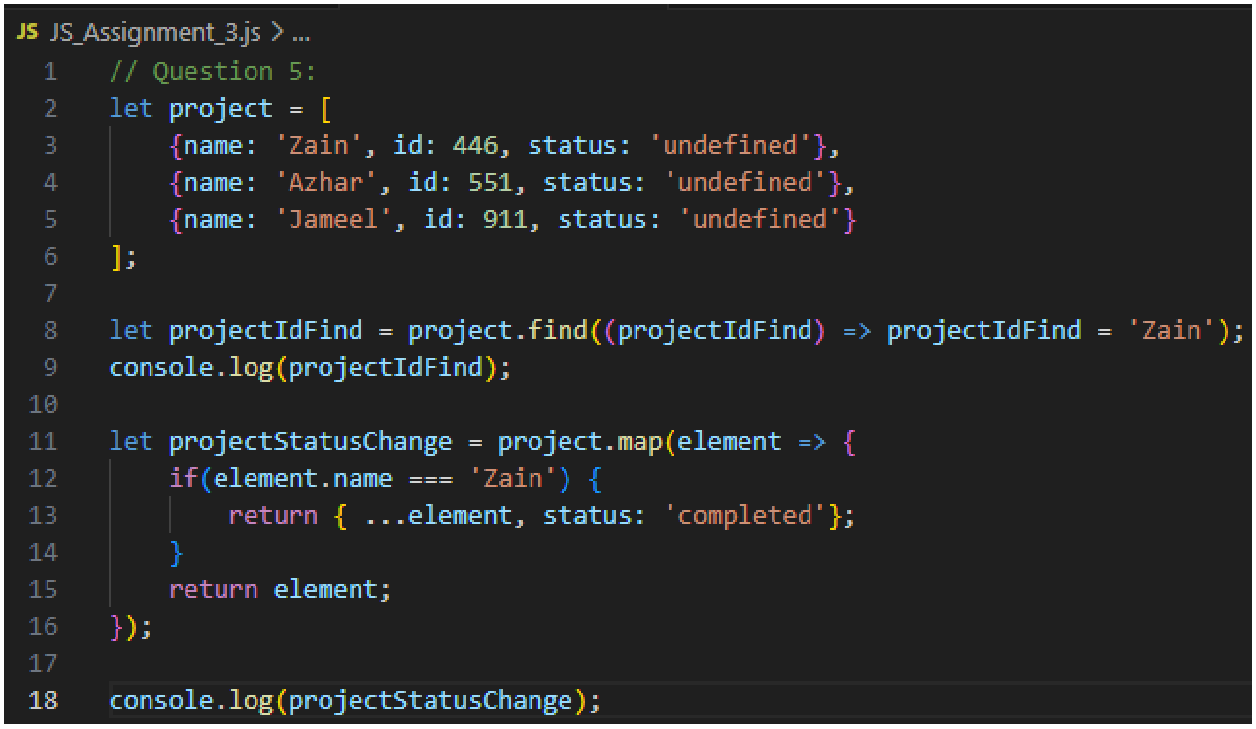


```
JS_Assignment_3.js > ...
// Question 4:
let arrOfEvenNumbers = [22, 54, 76, 98, 110, 80, 66];

let arrOfEvenNumbersReduced = arrOfEvenNumbers.reduce((acc, curr) => {
    return acc + curr;
}, 0);

console.log(arrOfEvenNumbersReduced);
```

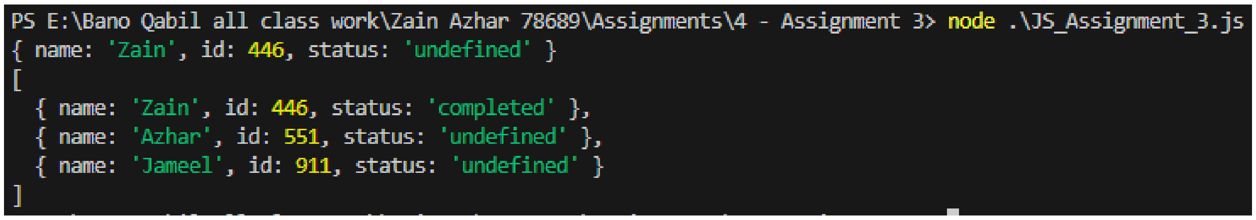


```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
506
```

Q5: 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'.

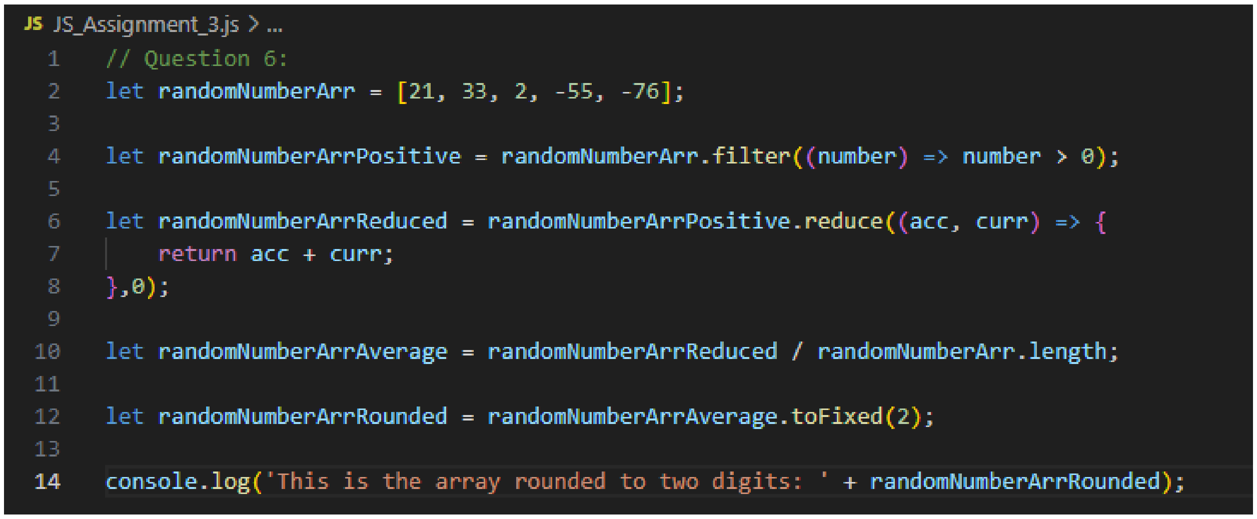


```
JS_Assignment_3.js > ...
// Question 5:
let project = [
    {name: 'Zain', id: 446, status: 'undefined'},
    {name: 'Azhar', id: 551, status: 'undefined'},
    {name: 'Jameel', id: 911, status: 'undefined'}
];

let projectIdFind = project.find((projectIdFind) => projectIdFind = 'Zain');
console.log(projectIdFind);

let projectStatusChange = project.map(element => {
    if(element.name === 'Zain') {
        return { ...element, status: 'completed'};
    }
    return element;
});

console.log(projectStatusChange);
```



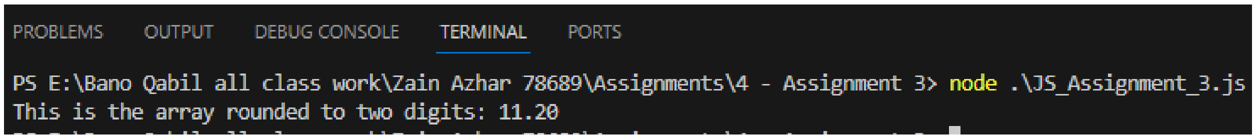
```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
{ name: 'Zain', id: 446, status: 'undefined' }
[
  { name: 'Zain', id: 446, status: 'completed' },
  { name: 'Azhar', id: 551, status: 'undefined' },
  { name: 'Jameel', id: 911, status: 'undefined' }
]
```

Q6: 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.



JS_Assignment_3.js > ...

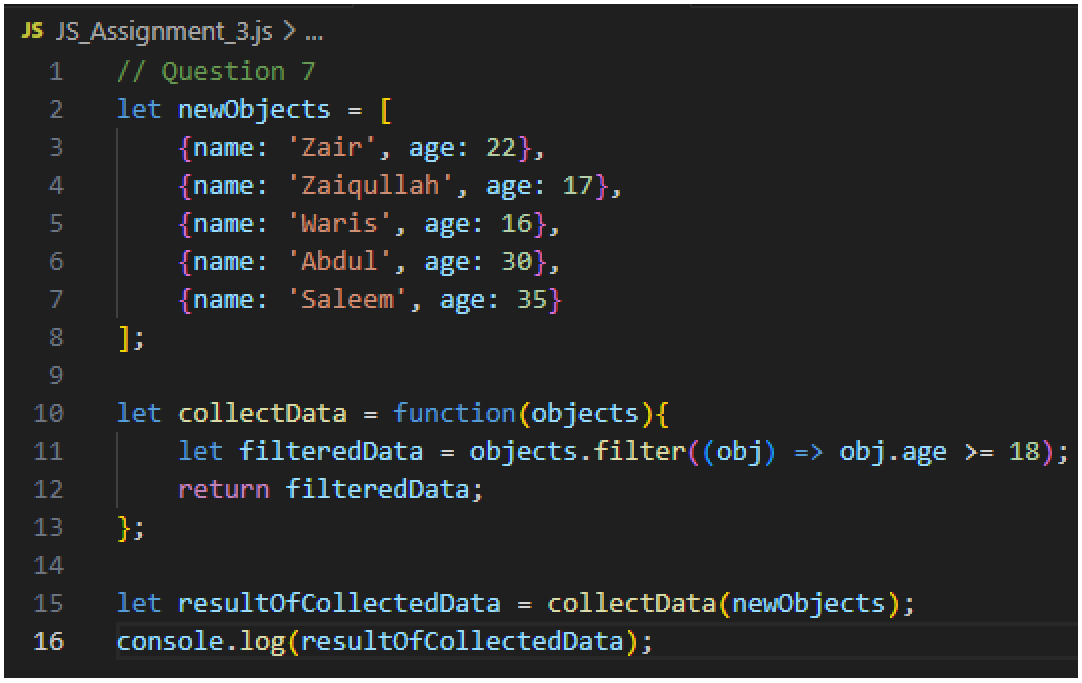
```javascript
// Question 6:
let randomNumberArr = [21, 33, 2, -55, -76];

let randomNumberArrPositive = randomNumberArr.filter((number) => number > 0);

let randomNumberArrReduced = randomNumberArrPositive.reduce((acc, curr) => {
    return acc + curr;
},0);

let randomNumberArrAverage = randomNumberArrReduced / randomNumberArr.length;

let randomNumberArrRounded = randomNumberArrAverage.toFixed(2);

console.log('This is the array rounded to two digits: ' + randomNumberArrRounded);
```



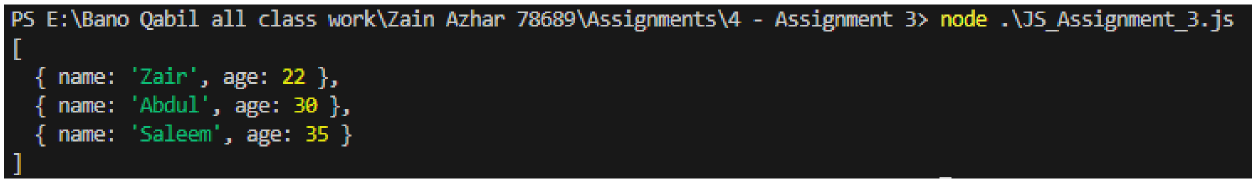
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
This is the array rounded to two digits: 11.20
```

Q7: 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.

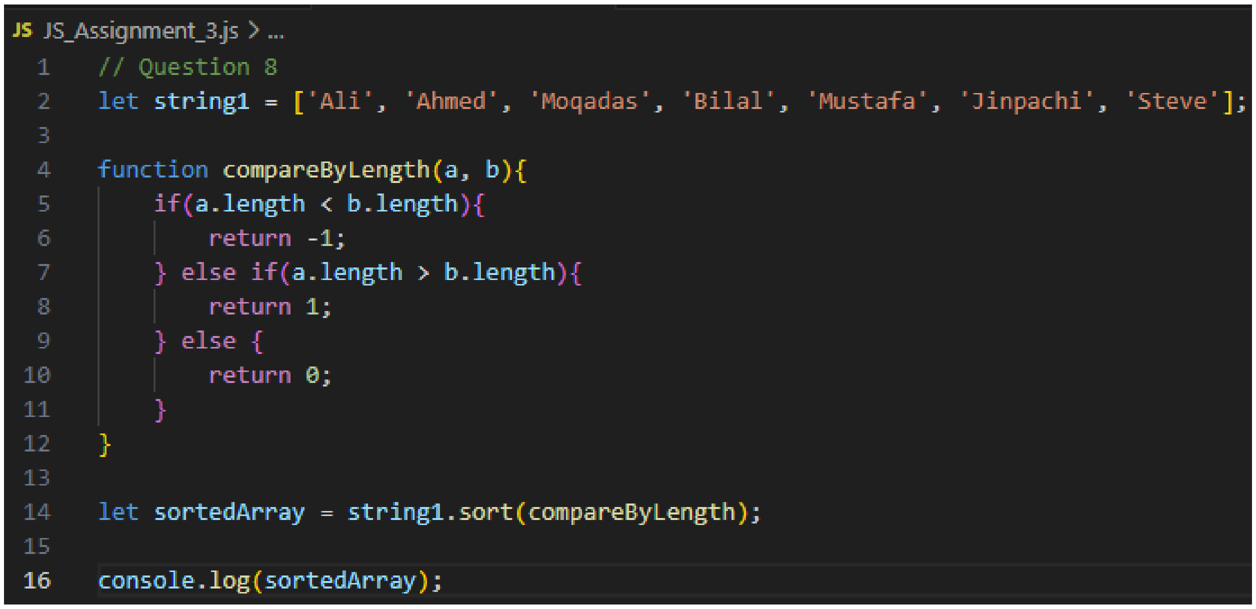


```
JS JS_Assignment_3.js > ...
// Question 7
let newObjects = [
    {name: 'Zair', age: 22},
    {name: 'Zaiqullah', age: 17},
    {name: 'Waris', age: 16},
    {name: 'Abdul', age: 30},
    {name: 'Saleem', age: 35}
];

let collectData = function(objects){
    let filteredData = objects.filter((obj) => obj.age >= 18);
    return filteredData;
};

let resultOfCollectedData = collectData(newObjects);
console.log(resultOfCollectedData);
```

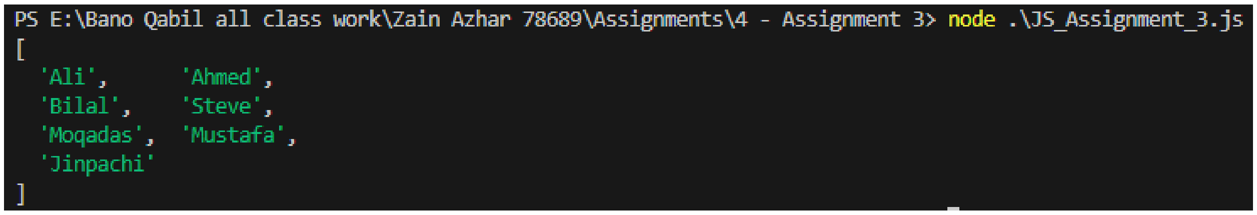


```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[
  { name: 'Zair', age: 22 },
  { name: 'Abdul', age: 30 },
  { name: 'Saleem', age: 35 }
]
```

Q8: 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.

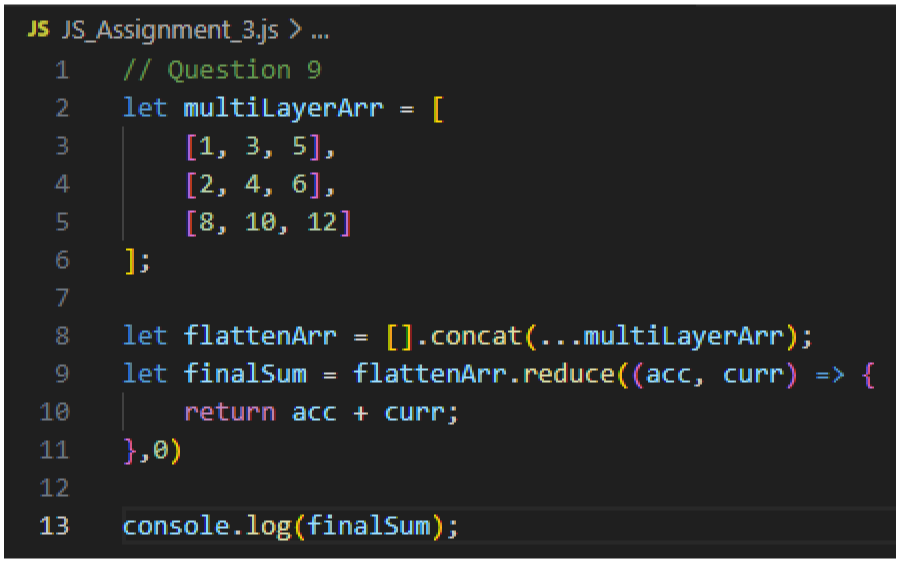

JS_Assignment_3.js > ...

```
// Question 8
let string1 = ['Ali', 'Ahmed', 'Moqadas', 'Bilal', 'Mustafa', 'Jinpachi', 'Steve'];

function compareByLength(a, b){
    if(a.length < b.length){
        return -1;
    } else if(a.length > b.length){
        return 1;
    } else {
        return 0;
    }
}

let sortedArray = string1.sort(compareByLength);

console.log(sortedArray);
```

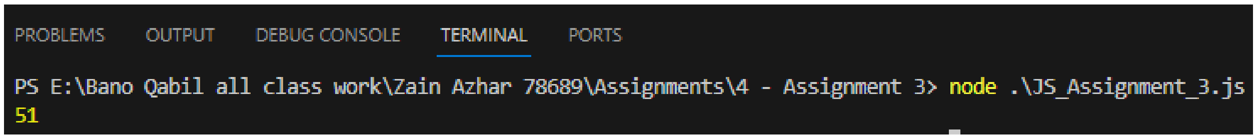


```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
[
  'Ali',      'Ahmed',
  'Bilal',    'Steve',
  'Moqadas',  'Mustafa',
  'Jinpachi'
]
```

Q9: 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.

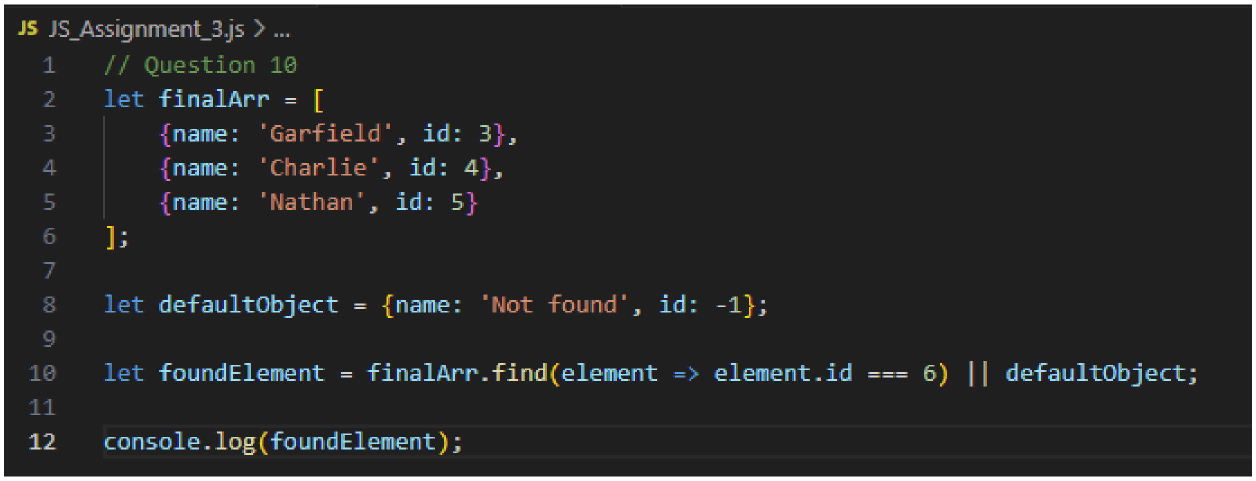

JS_Assignment_3.js > ...

```
// Question 9
let multiLayerArr = [
    [1, 3, 5],
    [2, 4, 6],
    [8, 10, 12]
];

let flattenArr = [].concat(...multiLayerArr);
let finalSum = flattenArr.reduce((acc, curr) => {
    return acc + curr;
},0)

console.log(finalSum);
```

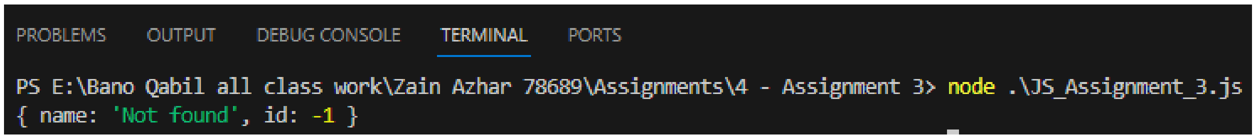


```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
51
```

Q10: Modify the `find` method to handle the scenario where the desired element is not found, returning a custom default object instead.



```javascript
// Question 10
let finalArr = [
    {name: 'Garfield', id: 3},
    {name: 'Charlie', id: 4},
    {name: 'Nathan', id: 5}
];

let defaultObject = {name: 'Not found', id: -1};

let foundElement = finalArr.find(element => element.id === 6) || defaultObject;

console.log(foundElement);
```



```
PS E:\Bano Qabil all class work\Zain Azhar 78689\Assignments\4 - Assignment 3> node .\JS_Assignment_3.js
{ name: 'Not found', id: -1 }
```