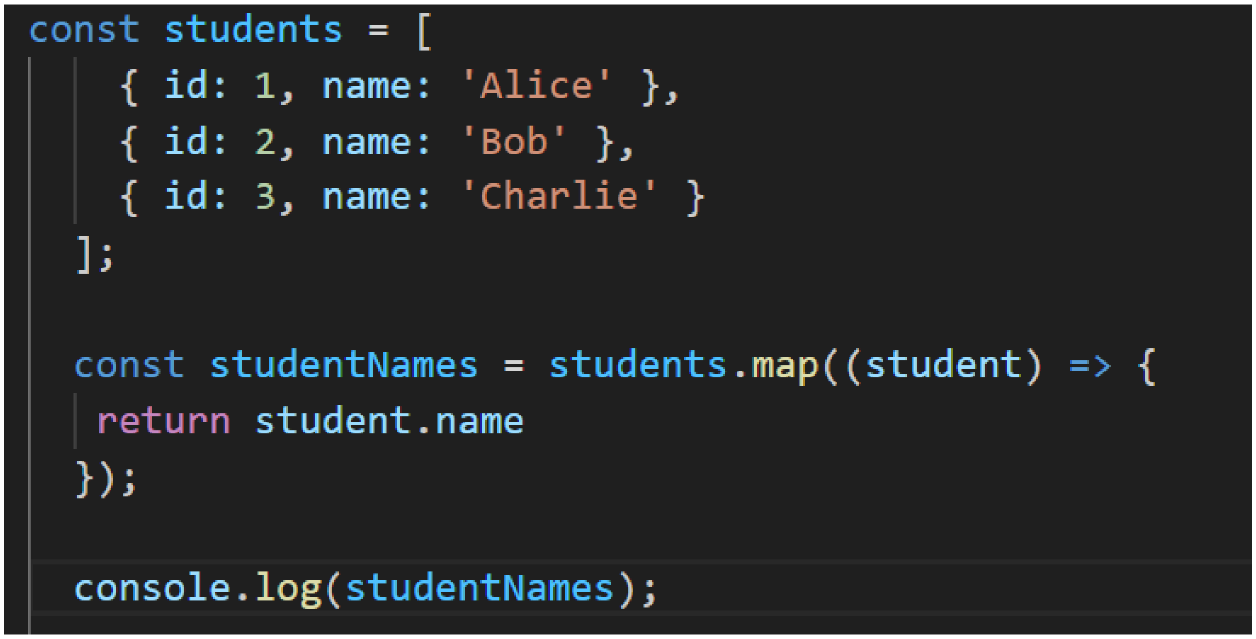
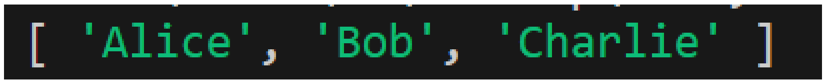
. **Map Method:**
- Q: How does the `map` method work in JavaScript, and can you provide an example of when you might use it to manipulate an array of objects?

Source Code



```
const students = [
    { id: 1, name: 'Alice' },
    { id: 2, name: 'Bob' },
    { id: 3, name: 'Charlie' }
  ];

  const studentNames = students.map((student) => {
   return student.name
  });

  console.log(studentNames);
```

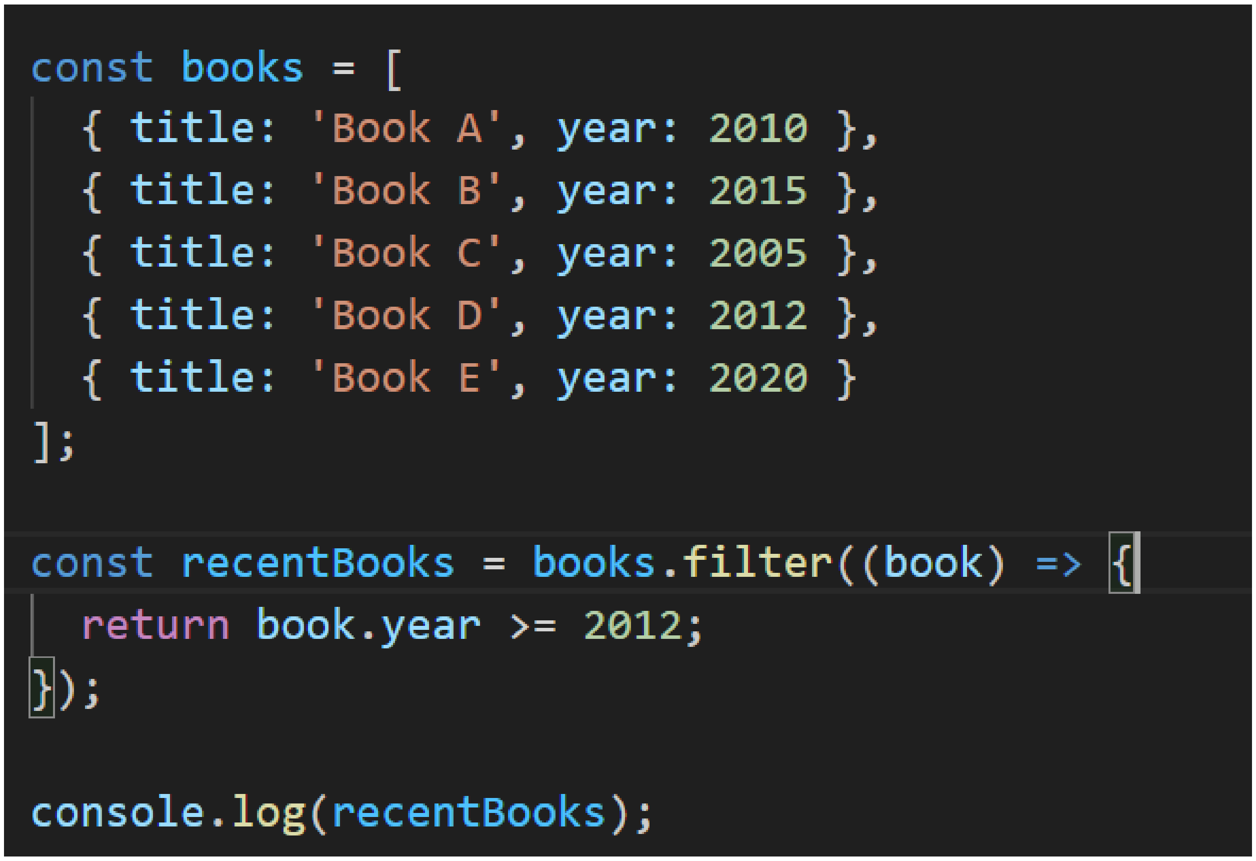
Output



```
[ 'Alice', 'Bob', 'Charlie' ]
```

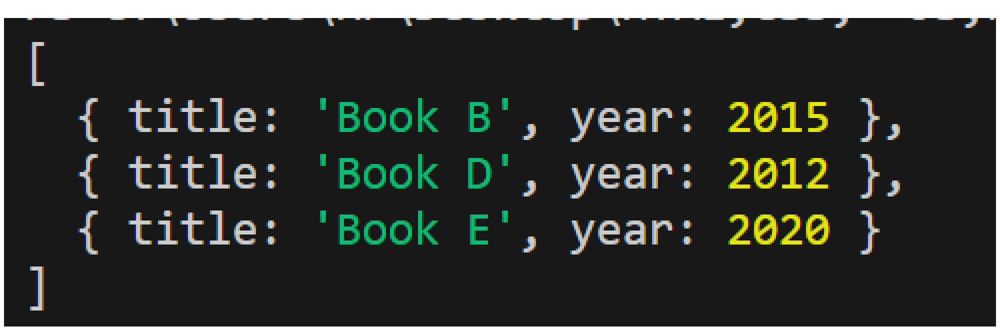
Filter Method:
- Q: Explain the purpose of the `filter` method. Provide an example where you use `filter` to extract elements from an array based on a specific condition.

Source Code



```
const books = [
  { title: 'Book A', year: 2010 },
  { title: 'Book B', year: 2015 },
  { title: 'Book C', year: 2005 },
  { title: 'Book D', year: 2012 },
  { title: 'Book E', year: 2020 }
];

const recentBooks = books.filter((book) => {
  return book.year >= 2012;
});

console.log(recentBooks);
```

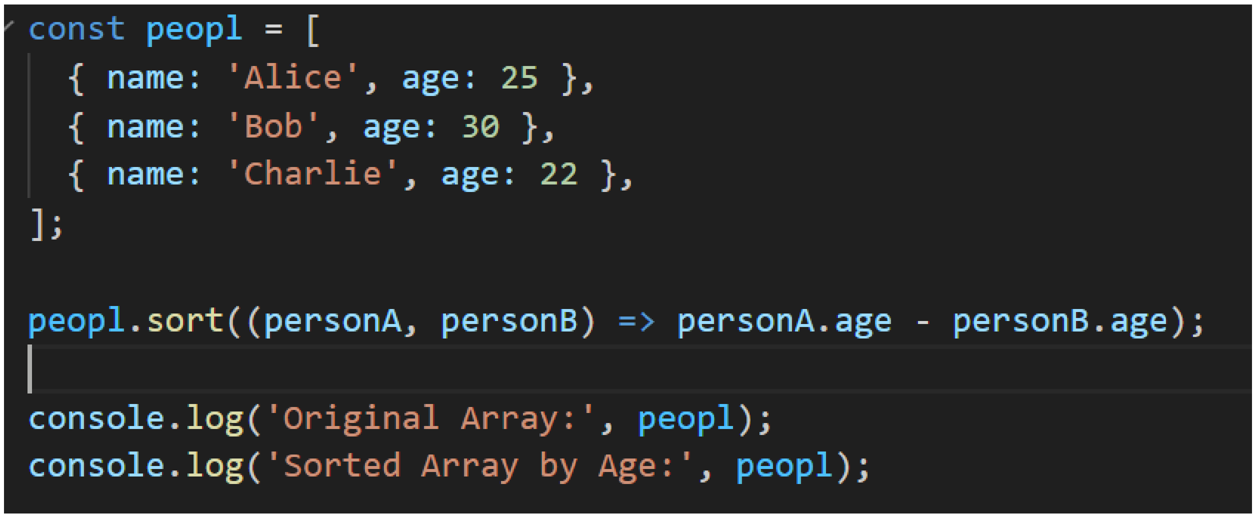
Output



```
[
  { title: 'Book B', year: 2015 },
  { title: 'Book D', year: 2012 },
  { title: 'Book E', year: 2020 }
]
```

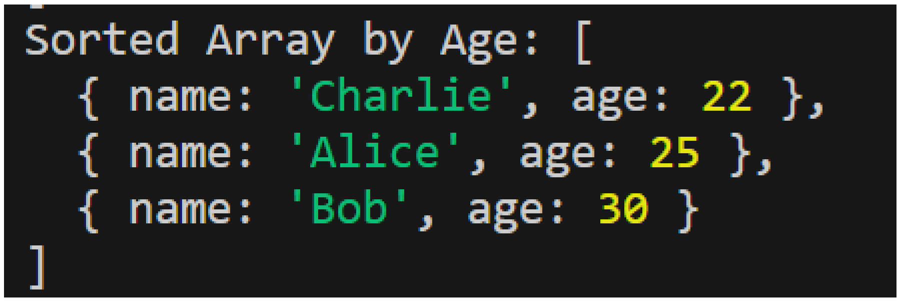
3. **Sort Method:**
- Q: Discuss the default behavior of the `sort` method for strings and numbers. How would you use a custom comparison function to sort an array of objects by a specific property?

Source Code



```
const peopl = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 22 },
];

peopl.sort((personA, personB) => personA.age - personB.age);

console.log('Original Array:', peopl);
console.log('Sorted Array by Age:', peopl);
```

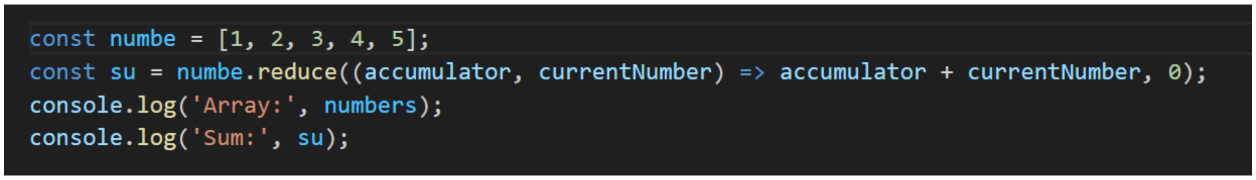
Output



```
Sorted Array by Age: [
  { name: 'Charlie', age: 22 },
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 }
]
```

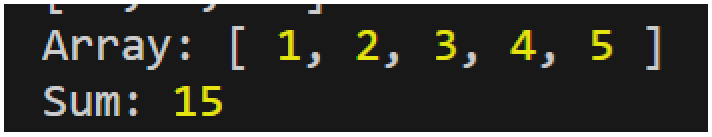
4. **Reduce Method:**
- Q: Describe the purpose of the `reduce` method and provide an example where you use it to compute a single value from an array of numbers.

Source Code



```
const numbe = [1, 2, 3, 4, 5];
const su = numbe.reduce((accumulator, currentNumber) => accumulator + currentNumber, 0);
console.log('Array:', numbers);
console.log('Sum:', su);
```

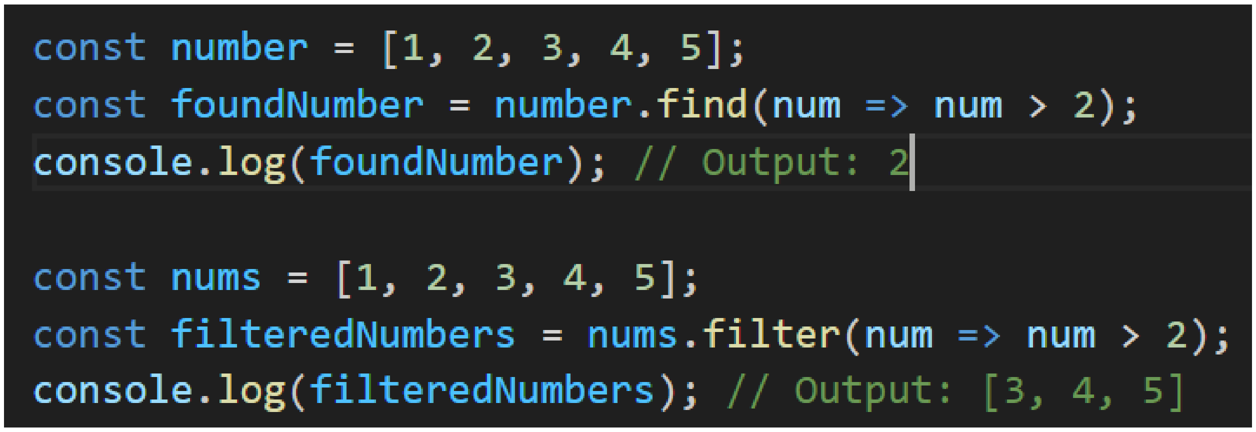
Output



```
Array: [ 1, 2, 3, 4, 5 ]
Sum: 15
```

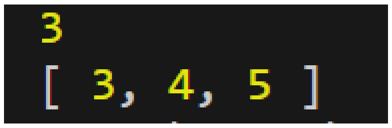
5. **Find Method:**
- Q: How does the `find` method differ from `filter`? Give an example of a scenario where using `find` is more appropriate than `filter`.

Source Code



```
const number = [1, 2, 3, 4, 5];
const foundNumber = number.find(num => num > 2);
console.log(foundNumber); // Output: 2

const nums = [1, 2, 3, 4, 5];
const filteredNumbers = nums.filter(num => num > 2);
console.log(filteredNumbers); // Output: [3, 4, 5]
```

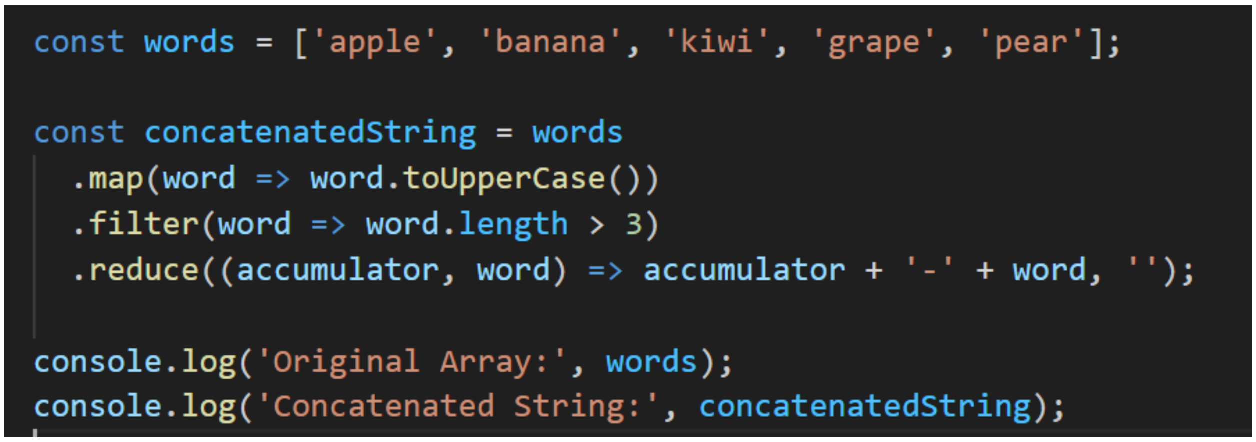
Output



```
3
[ 3, 4, 5 ]
```

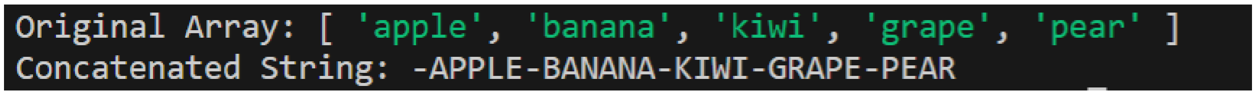
. **Combining Methods:**
- Q: Create a chain of array methods (`map`, `filter`, `reduce`, etc.) to transform an array of strings into a single concatenated string with a specific condition.

Source Code



```
const words = ['apple', 'banana', 'kiwi', 'grape', 'pear'];

const concatenatedString = words
  .map(word => word.toUpperCase())
  .filter(word => word.length > 3)
  .reduce((accumulator, word) => accumulator + '-' + word, '');

console.log('Original Array:', words);
console.log('Concatenated String:', concatenatedString);
```

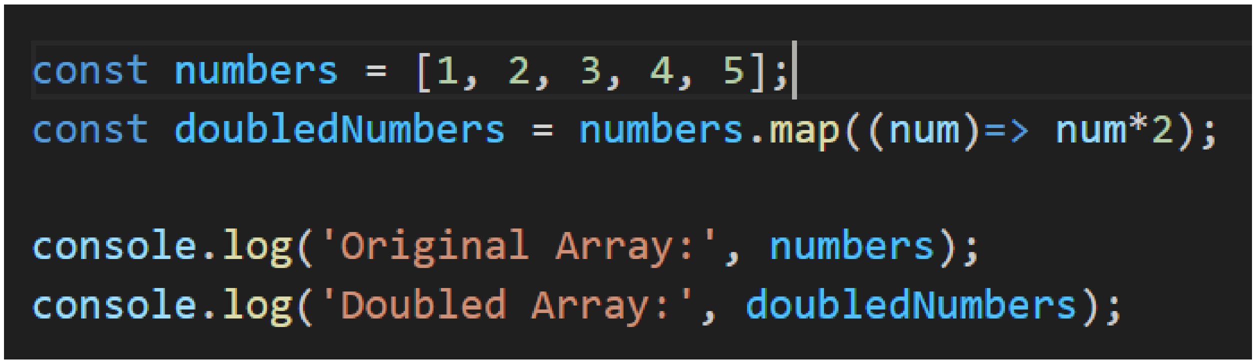
Output



```
Original Array: [ 'apple', 'banana', 'kiwi', 'grape', 'pear' ]
Concatenated String: -APPLE-BANANA-KIWI-GRAPE-PEAR
```

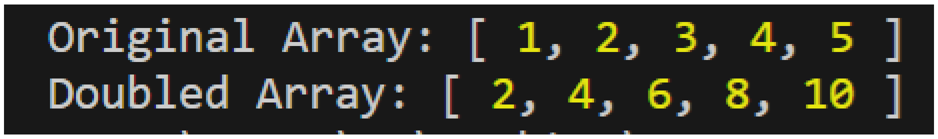
Nested Array Operations:
- Q: Given an array of arrays containing numbers, use a combination of array methods to flatten the structure and then calculate the sum of all the numbers.

Source Code



```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map((num)=> num*2);

console.log('Original Array:', numbers);
console.log('Doubled Array:', doubledNumbers);
```

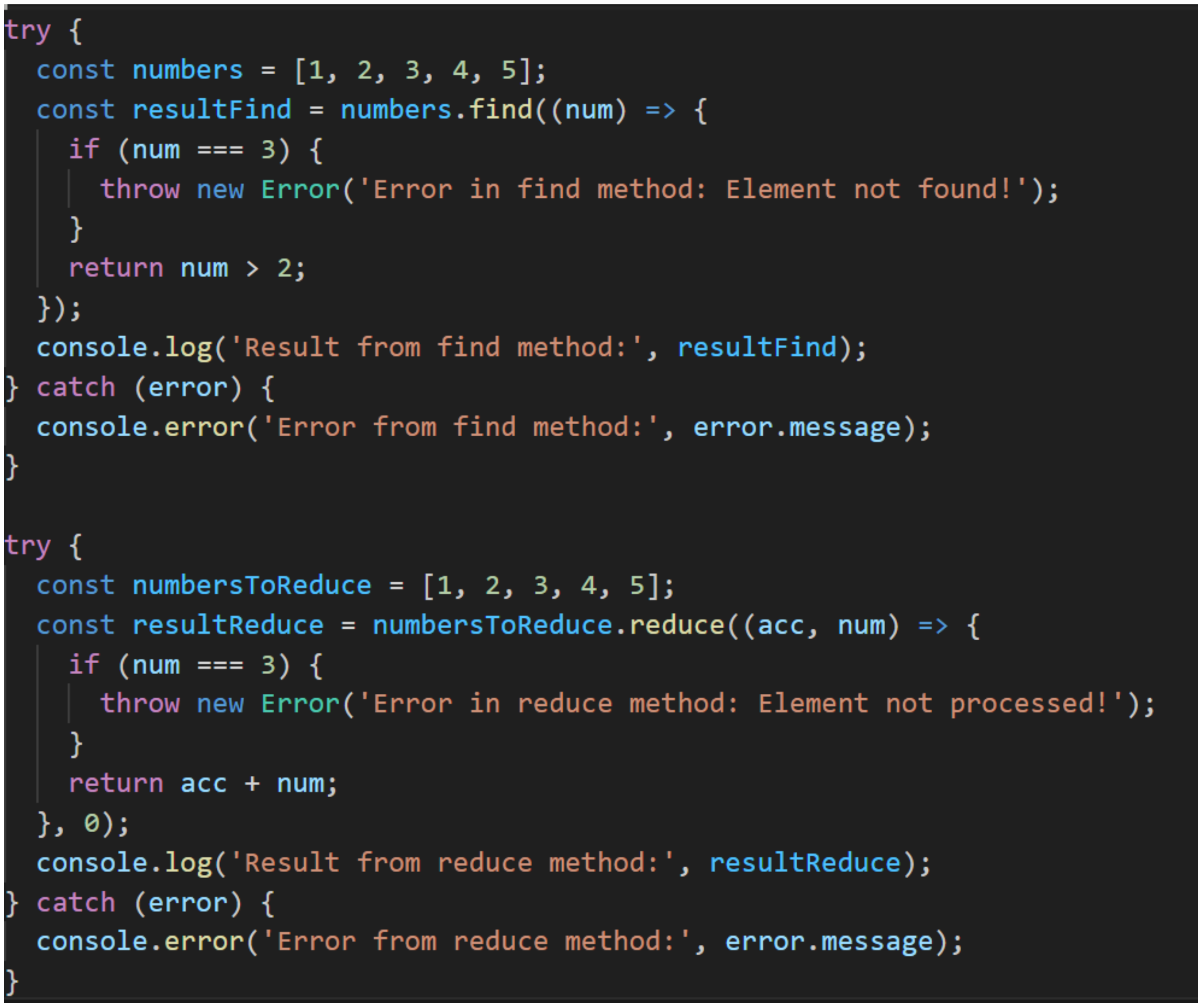
Output



```
Original Array: [ 1, 2, 3, 4, 5 ]
Doubled Array: [ 2, 4, 6, 8, 10 ]
```

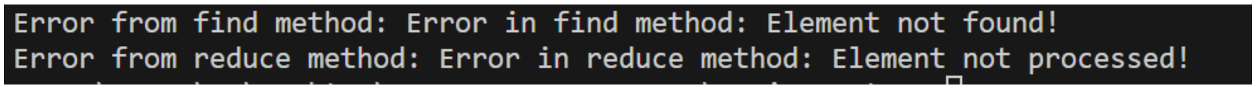
Error Handling:
- Q: How would you handle potential errors when using array methods like `find` or `reduce`? Provide an example of error handling in such a scenario.

Source Code



```
try {
  const numbers = [1, 2, 3, 4, 5];
  const resultFind = numbers.find((num) => {
    if (num === 3) {
      throw new Error('Error in find method: Element not found!');
    }
    return num > 2;
  });
  console.log('Result from find method:', resultFind);
} catch (error) {
  console.error('Error from find method:', error.message);
}

try {
  const numbersToReduce = [1, 2, 3, 4, 5];
  const resultReduce = numbersToReduce.reduce((acc, num) => {
    if (num === 3) {
      throw new Error('Error in reduce method: Element not processed!');
    }
    return acc + num;
  }, 0);
  console.log('Result from reduce method:', resultReduce);
} catch (error) {
  console.error('Error from reduce method:', error.message);
}
```

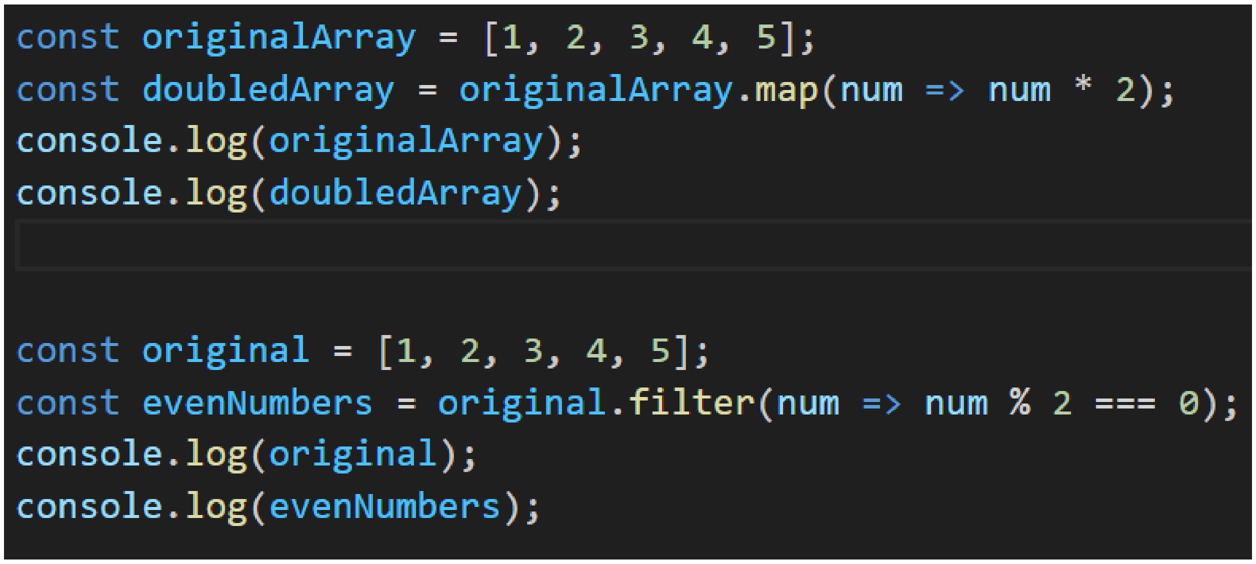
Output



```
Error from find method: Error in find method: Element not found!
Error from reduce method: Error in reduce method: Element not processed!
```

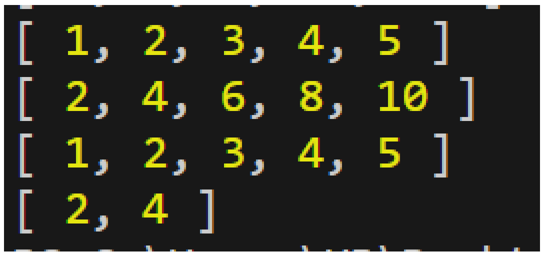
Immutable Operations:
- Q: Discuss the importance of immutability when working with array methods. Demonstrate how you would perform immutable operations using methods like `map` or `filter`.

Source Code



```
const originalArray = [1, 2, 3, 4, 5];
const doubledArray = originalArray.map(num => num * 2);
console.log(originalArray);
console.log(doubledArray);


const original = [1, 2, 3, 4, 5];
const evenNumbers = original.filter(num => num % 2 === 0);
console.log(original);
console.log(evenNumbers);
```

Output



```
[ 1, 2, 3, 4, 5 ]
[ 2, 4, 6, 8, 10 ]
[ 1, 2, 3, 4, 5 ]
[ 2, 4 ]
```

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

<u>ForEach</u>

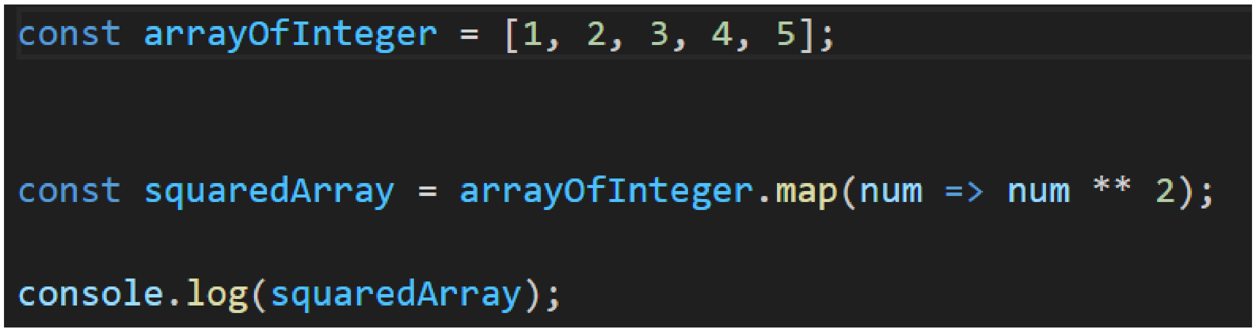
forEach without creating a new Array

<u>Map</u>

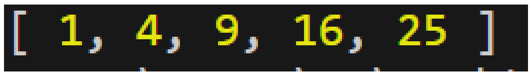
Map create new Array

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

Source Code

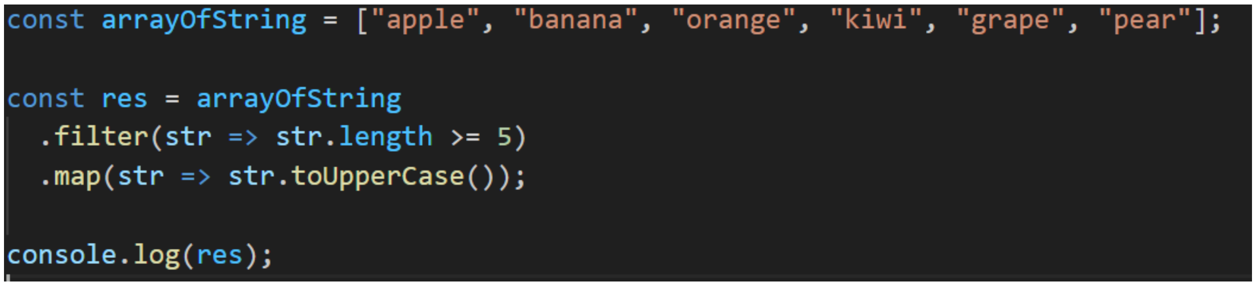

```
const arrayOfInteger = [1, 2, 3, 4, 5];


const squaredArray = arrayOfInteger.map(num => num ** 2);

console.log(squaredArray);
```

Output

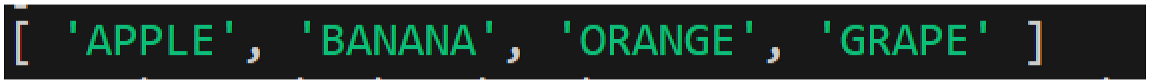

```
[ 1, 4, 9, 16, 25 ]
```

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.

Source Code

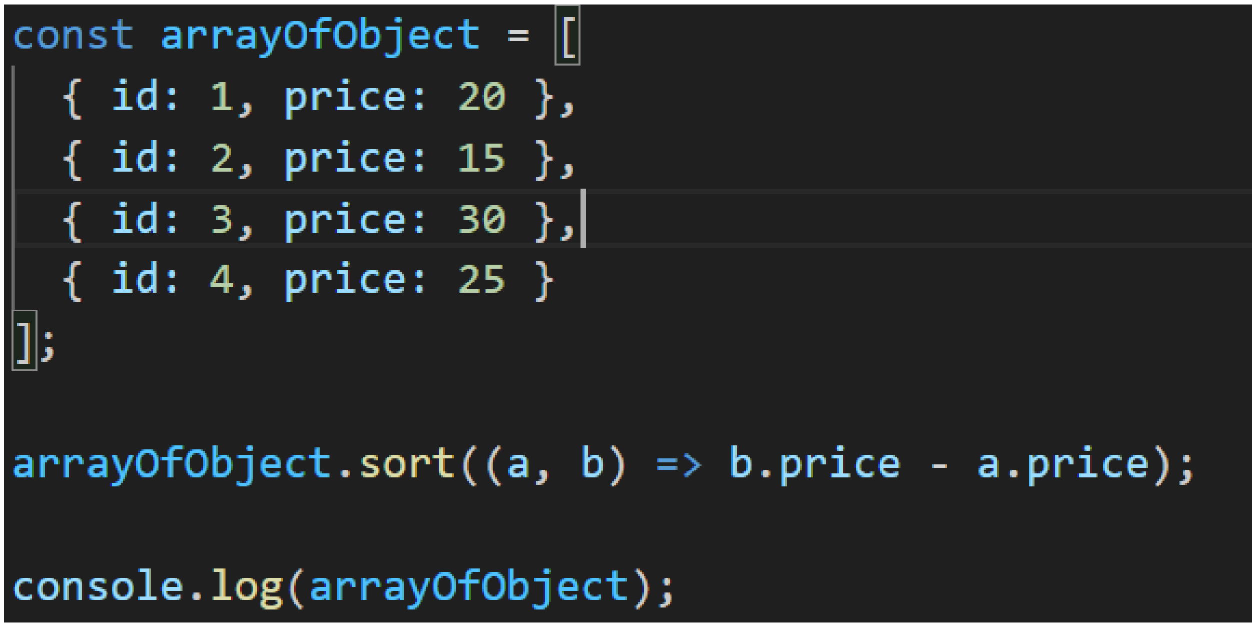

```
const arrayOfString = ["apple", "banana", "orange", "kiwi", "grape", "pear"];

const res = arrayOfString
  .filter(str => str.length >= 5)
  .map(str => str.toUpperCase());

console.log(res);
```

Output


```
[ 'APPLE', 'BANANA', 'ORANGE', 'GRAPE' ]
```

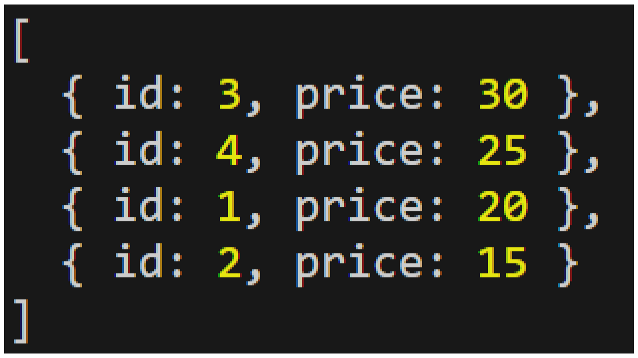
3. **Sorting Objects:**
- Q: Given an array of objects with a 'price' property, use the `sort` method to arrange them in descending order based on their price

Source Code



```
const arrayOfObject = [
  { id: 1, price: 20 },
  { id: 2, price: 15 },
  { id: 3, price: 30 },
  { id: 4, price: 25 }
];

arrayOfObject.sort((a, b) => b.price - a.price);

console.log(arrayOfObject);
```

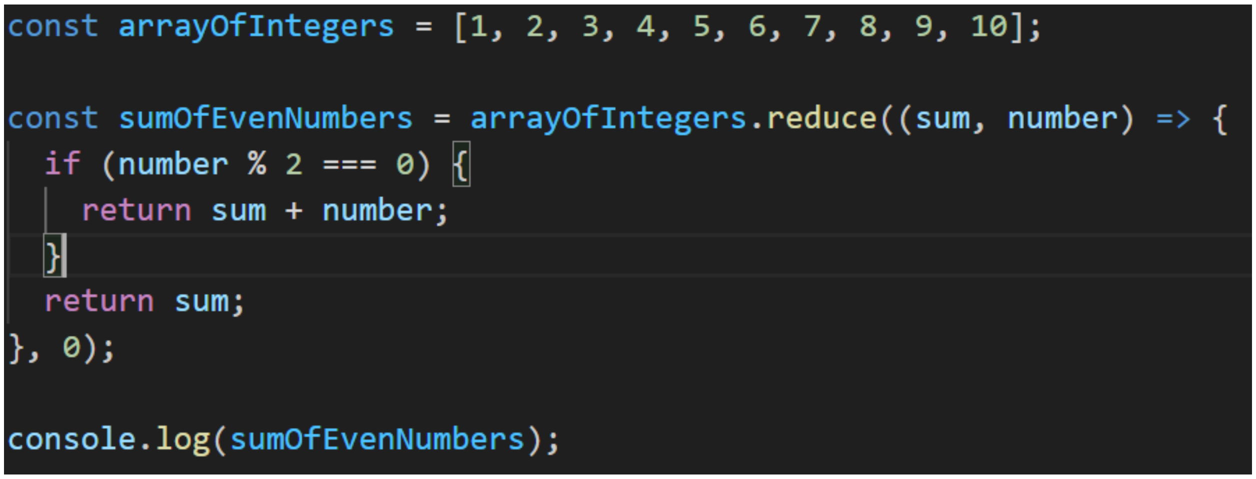
Output



```
[
  { id: 3, price: 30 },
  { id: 4, price: 25 },
  { id: 1, price: 20 },
  { id: 2, price: 15 }
]
```

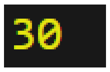
4. **Reduce for Aggregation:**
- Q: Use the `reduce` method to find the total sum of all even numbers in an array of integers

Source Code



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

const sumOfEvenNumbers = arrayOfIntegers.reduce((sum, number) => {
  if (number % 2 === 0) {
    return sum + number;
  }
  return sum;
}, 0);

console.log(sumOfEvenNumbers);
```

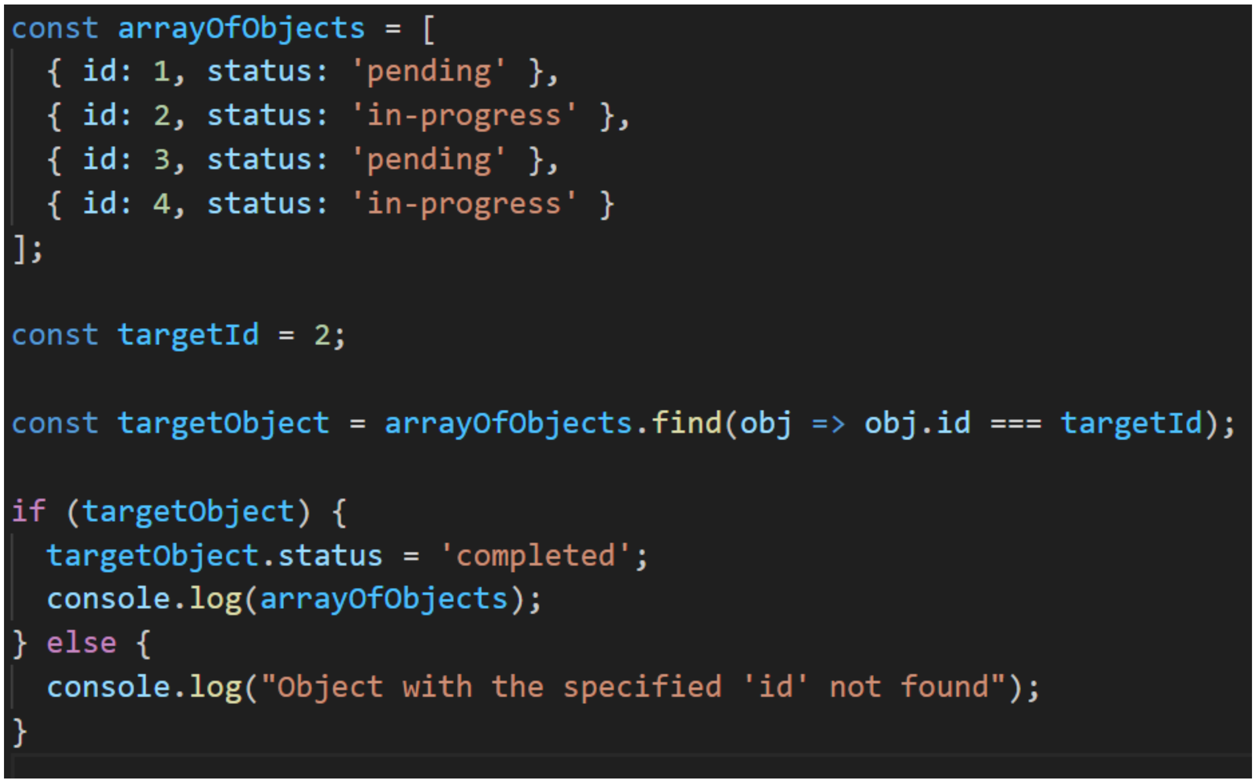
Output



```
30
```

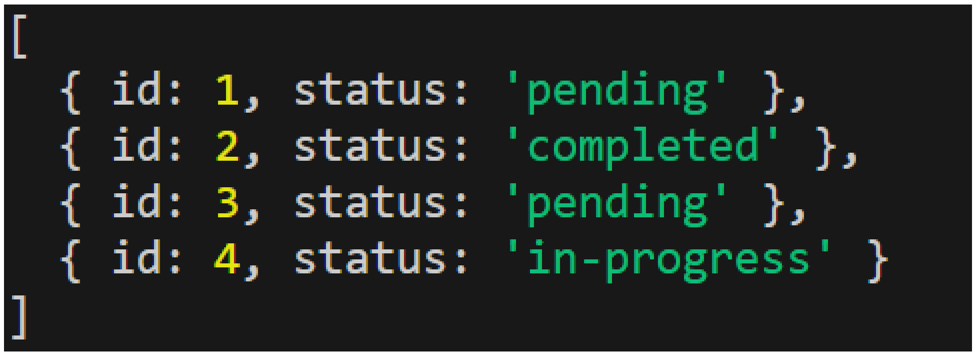
5. **Find and Modify:**
- Q: Given an array of objects with 'id' properties, use the `find` method to locate an object with a specific 'id' and update its 'status' property to 'completed'.

Source Code



```javascript
const arrayOfObjects = [
  { id: 1, status: 'pending' },
  { id: 2, status: 'in-progress' },
  { id: 3, status: 'pending' },
  { id: 4, status: 'in-progress' }
];

const targetId = 2;

const targetObject = arrayOfObjects.find(obj => obj.id === targetId);

if (targetObject) {
  targetObject.status = 'completed';
  console.log(arrayOfObjects);
} else {
  console.log("Object with the specified 'id' not found");
}
```

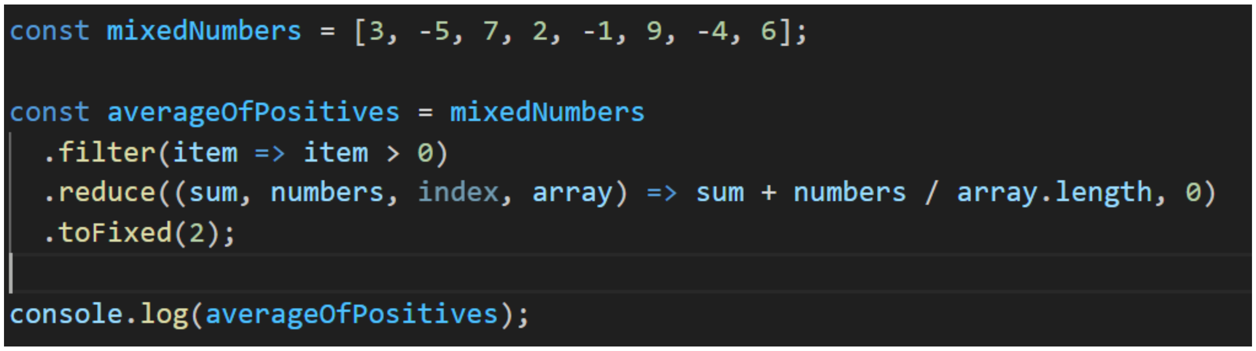
Output



```
[
  { id: 1, status: 'pending' },
  { id: 2, status: 'completed' },
  { id: 3, status: 'pending' },
  { id: 4, status: 'in-progress' }
]
```

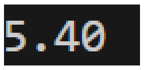
6. **Chaining Methods:**
- Q: Create a chain of array methods to find the average of all positive numbers in an array of mixed integers and return the result rounded to two decimal places.

Source Code



```
const mixedNumbers = [3, -5, 7, 2, -1, 9, -4, 6];

const averageOfPositives = mixedNumbers
  .filter(item => item > 0)
  .reduce((sum, numbers, index, array) => sum + numbers / array.length, 0)
  .toFixed(2);

console.log(averageOfPositives);
```

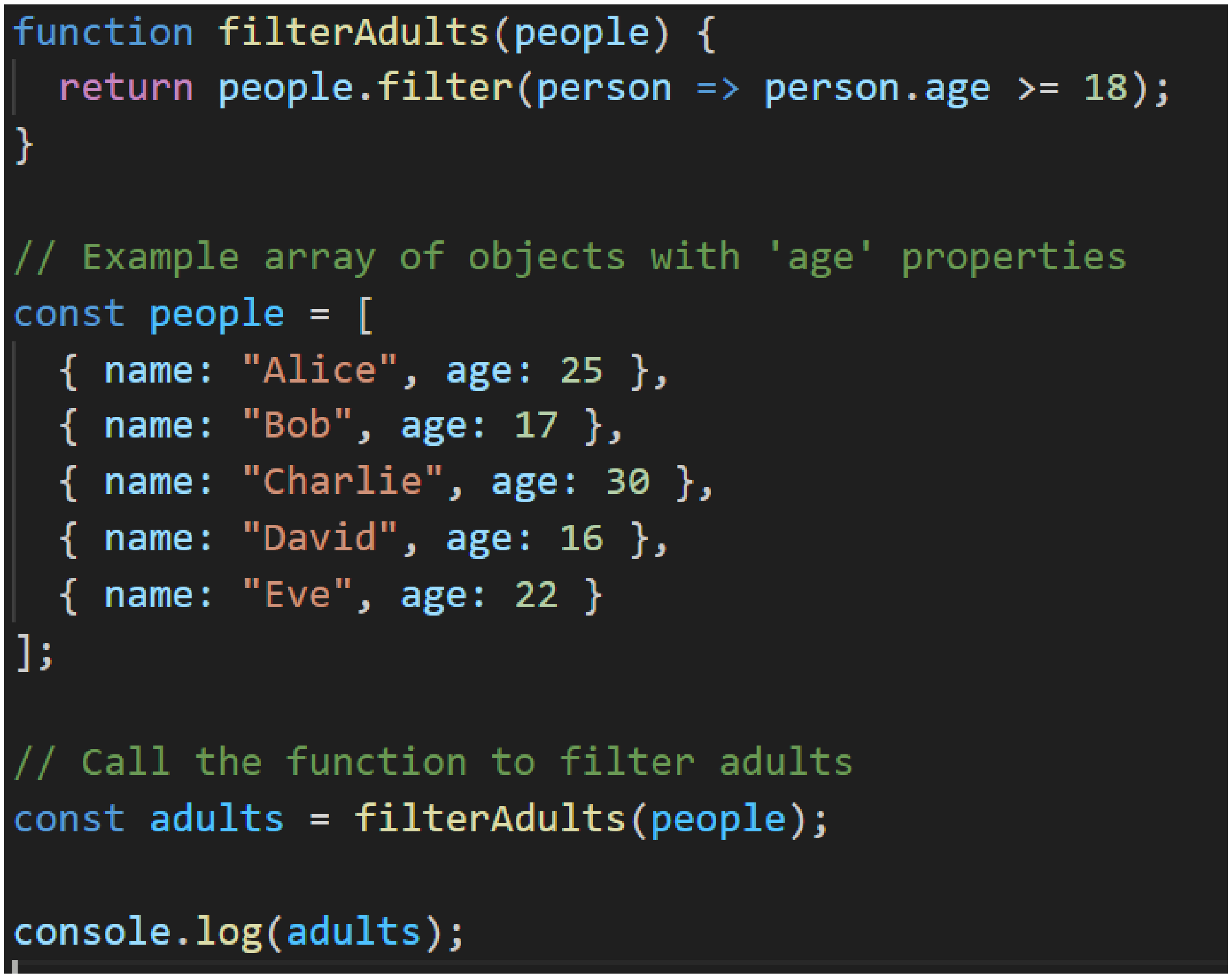
Output



```
5.40
```

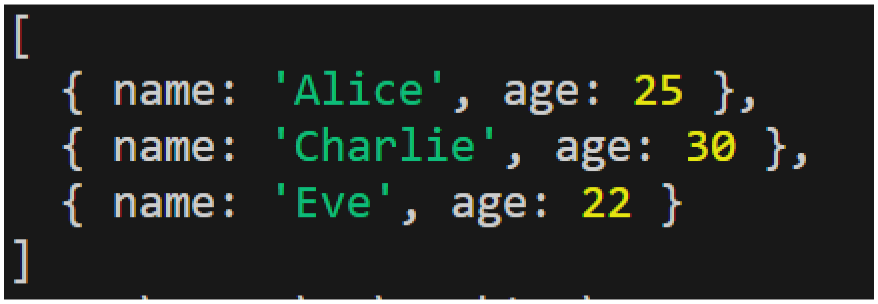
7. **Conditional Filtering**
- Q: Implement a function that takes an array of objects with 'age' properties and returns an array of those who are adults (age 18 and above) using the `filter` method.

Source Code



```
function filterAdults(people) {
  return people.filter(person => person.age >= 18);
}

// Example array of objects with 'age' properties
const people = [
  { name: "Alice", age: 25 },
  { name: "Bob", age: 17 },
  { name: "Charlie", age: 30 },
  { name: "David", age: 16 },
  { name: "Eve", age: 22 }
];

// Call the function to filter adults
const adults = filterAdults(people);

console.log(adults);
```

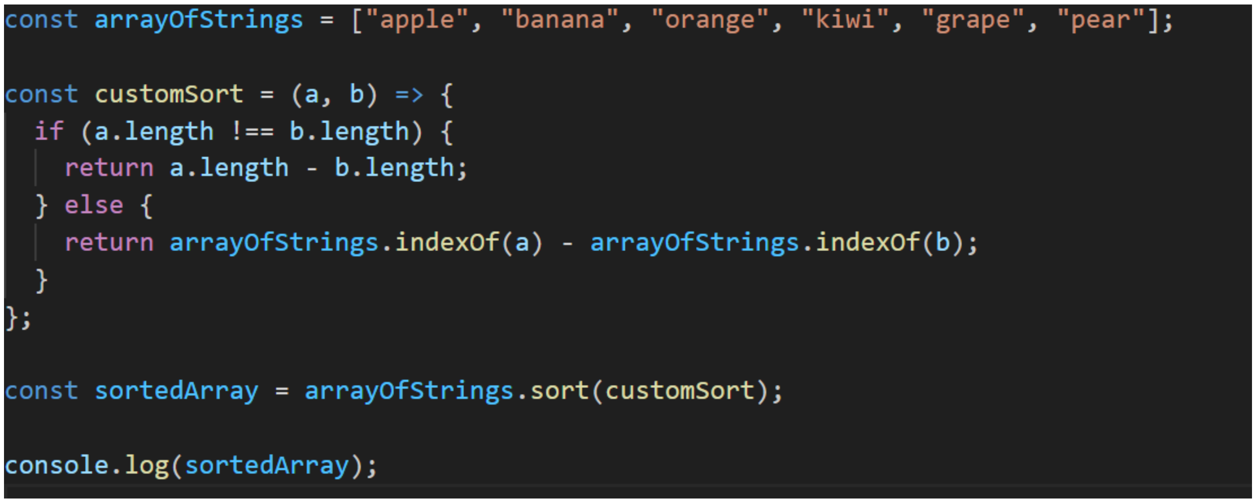
Output



```
[
  { name: 'Alice', age: 25 },
  { name: 'Charlie', age: 30 },
  { name: 'Eve', age: 22 }
]
```

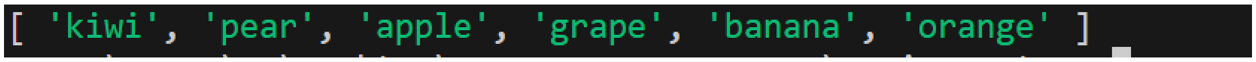
8. **Advanced Sorting:**
- Q: Sort an array of strings based on their lengths in ascending order. If two strings have the same length, maintain their relative order in the sorted array.

Source Code



```
const arrayOfStrings = ["apple", "banana", "orange", "kiwi", "grape", "pear"];

const customSort = (a, b) => {
  if (a.length !== b.length) {
    return a.length - b.length;
  } else {
    return arrayOfStrings.indexOf(a) - arrayOfStrings.indexOf(b);
  }
};

const sortedArray = arrayOfStrings.sort(customSort);

console.log(sortedArray);
```

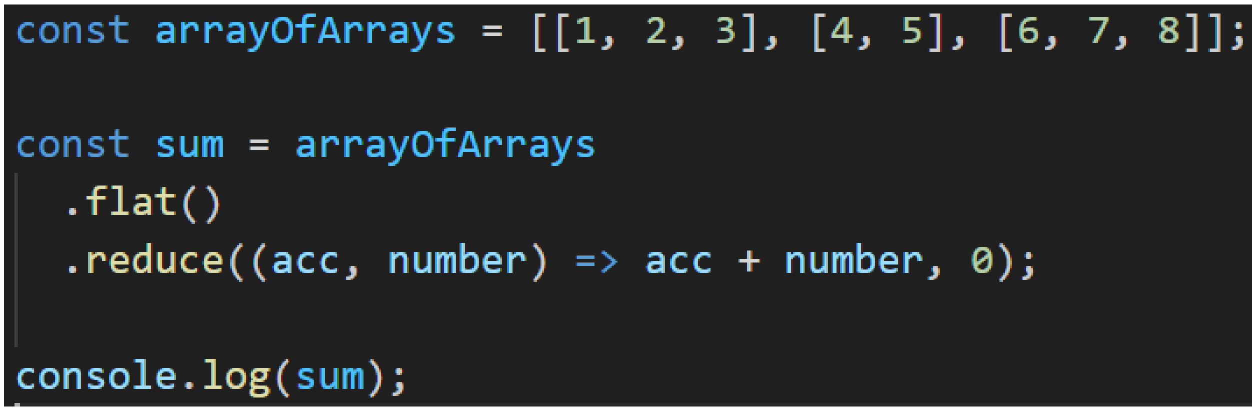
Output



```
[ 'kiwi', 'pear', 'apple', 'grape', 'banana', 'orange' ]
```

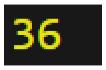
9. **Nested Array Operations:**
- Q: Given an array of arrays containing numbers, use a combination of array methods to flatten the structure and then calculate the sum of all the numbers.

Source Code



```
const arrayOfArrays = [[1, 2, 3], [4, 5], [6, 7, 8]];

const sum = arrayOfArrays
  .flat()
  .reduce((acc, number) => acc + number, 0);

console.log(sum);
```

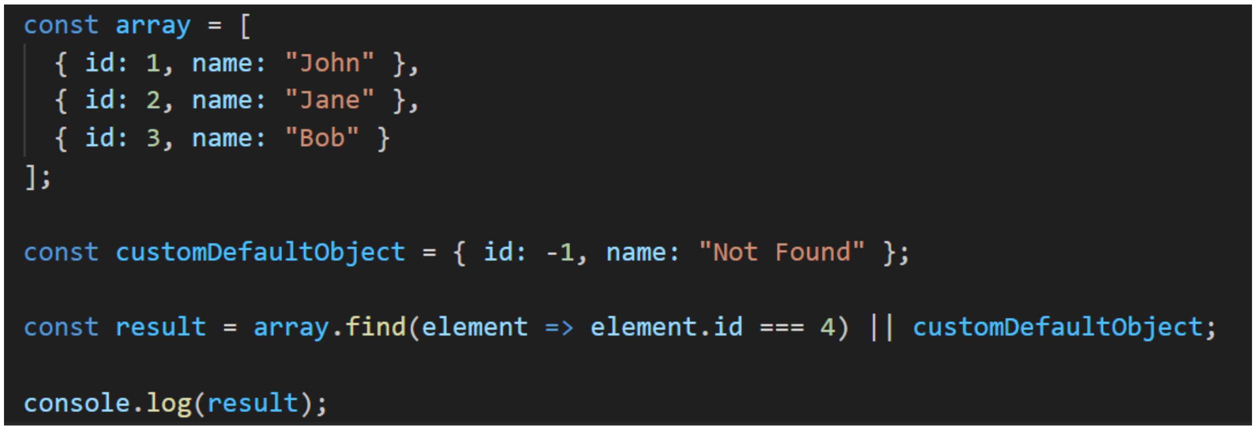
Output



```
36
```

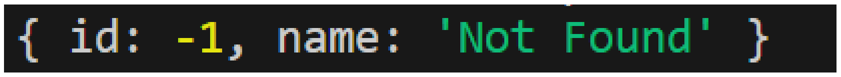
10. **Error Handling with Find**
- Q: Modify the `find` method to handle the scenario where the desired element is not found, returning a custom default object instead.

Source Code



```
const array = [
  { id: 1, name: "John" },
  { id: 2, name: "Jane" },
  { id: 3, name: "Bob" }
];

const customDefaultObject = { id: -1, name: "Not Found" };

const result = array.find(element => element.id === 4) || customDefaultObject;

console.log(result);
```

Output



```
{ id: -1, name: 'Not Found' }
```