[**Map method** 5](#_Toc152161347)

[**Exercise 1: Calculate Customer Score Using map()** 5](#_Toc152161348)

[Exercise 2: Normalize Product Prices Using map() 9](#_Toc152161349)

[Exercise 3: Convert Object Array to Array of Values 13](#_Toc152161350)

[Exercise 4: Calculate Total Price with Tax 15](#_Toc152161351)

[Exercise 5: Convert Minutes to Hours and Minutes 15](#_Toc152161352)

[**Exercise 6: Capitalize First Letter of Each Word** 16](#_Toc152161353)

[**Exercise 7: Convert RGB to Hex** 18](#_Toc152161354)

[**Exercise 8: Calculate Distance Between Coordinates** 21](#_Toc152161355)

[**Reduce method** 24](#_Toc152161356)

[**Exercise 1: Calculate the total inventory value for all products using reduce().** 24](#_Toc152161357)

[**Exercise 2: Calculate Overall Customer Satisfaction Score Using reduce().** 27](#_Toc152161358)

[**Exercise 3: Calculate Average Rating for Each Category** 29](#_Toc152161359)

[Exercise 4: Find Most Frequent Words in Text 33](#_Toc152161360)

[Exercise 5. Calculate Cumulative Monthly Sales 35](#_Toc152161361)

[Exercise 6. Group Data by Multiple Criteria 38](#_Toc152161362)

[Exercise 7. Compute Weighted Average 41](#_Toc152161363)

[Exercise 8. Flatten Nested Data Structures 44](#_Toc152161364)

[filter() 46](#_Toc152161365)

[Exercise 1: Filter Out Inactive Users 46](#_Toc152161366)

[Exercise 2: Filter Products Above a Price Range 49](#_Toc152161367)

[Exercise 3: Filter Out Unpublished Articles 51](#_Toc152161368)

[Exercise 4: Filter Out Users Not in a Given Country 54](#_Toc152161369)

[forEach() 56](#_Toc152161370)

[Exercise 1: Print User Emails 56](#_Toc152161371)

[Exercise 2: Update Stock Quantity 58](#_Toc152161372)

[Exercise 3: Send Notification to Users 61](#_Toc152161373)

[Exercise 4: Calculate Total Sales 63](#_Toc152161374)

[some() 65](#_Toc152161375)

[Exercise 1: Check for VIP Users 65](#_Toc152161376)

[Exercise 2: Check for Out of Stock Products 67](#_Toc152161377)

[Exercise 3: Check for Users from a Specific Country 69](#_Toc152161378)

[every() 73](#_Toc152161379)

[Exercise 1: Check if All Users are Active 73](#_Toc152161380)

[Exercise 2: Check if All Products are In Stock 75](#_Toc152161381)

[Exercise 3: Check if All Users are Above 18 77](#_Toc152161382)

[charAt() 79](#_Toc152161383)

[Exercise 1: Extract Initials from Name 79](#_Toc152161384)

[Exercise 2: Check First Letter of a Username 81](#_Toc152161385)

[Exercise 3: Extract File Extension 83](#_Toc152161386)

[indexOf() 84](#_Toc152161387)

[Exercise 1: Check if Email Contains '@' Symbol 84](#_Toc152161388)

[Exercise 2: Find Position of Substring 86](#_Toc152161389)

[Exercise 3: Check Protocol of a URL 87](#_Toc152161390)

[lastIndexOf() 89](#_Toc152161391)

[Exercise 1: Find Last Occurrence of a Word 89](#_Toc152161392)

[Exercise 2: Check File Extension 91](#_Toc152161393)

[Exercise 3: Extract Domain from Email 92](#_Toc152161394)

[slice() 94](#_Toc152161395)

[Exercise 1: Extract Username from Email 94](#_Toc152161396)

[Exercise 2: Get First Name from Full Name 96](#_Toc152161397)

[Exercise 3: Extract Protocol from URL 98](#_Toc152161398)

[split() 99](#_Toc152161399)

[Exercise 1: Split Sentence into Words 99](#_Toc152161400)

[**Exercise 2: Split CSV Line** 102](#_Toc152161401)

[**Exercise 3: Split URL Parameters** 105](#_Toc152161402)

[replace() 108](#_Toc152161403)

[**Exercise 1: Replace Placeholder** 108](#_Toc152161404)

[**Exercise 2: Replace Spaces with Dashes** 111](#_Toc152161405)

[**Exercise 3: Replace Domain in Email Address** 114](#_Toc152161406)

[trim() 117](#_Toc152161407)

[**Exercise 1: Trim Whitespace** 117](#_Toc152161408)

[**Exercise 2: Log Trimmed Input** 119](#_Toc152161409)

[**Exercise 3: Validate Non-empty Input** 122](#_Toc152161410)

[toUpperCase() 124](#_Toc152161411)

[**Exercise 1: Convert to Uppercase** 124](#_Toc152161412)

[**Exercise 2: Uppercase First Letter** 127](#_Toc152161413)

[**Exercise 3: Uppercase Log Level** 130](#_Toc152161414)

[toLowerCase() 133](#_Toc152161415)

[**Exercise 1: Convert to Lowercase** 133](#_Toc152161416)

[**Exercise 2: Lowercase Email Address** 136](#_Toc152161417)

[**Exercise 3: Lowercase User Input** 139](#_Toc152161418)

[Object.assign 147](#_Toc152161419)

[**Exercise 1: Merge Objects** 147](#_Toc152161420)

[Exercise 2: Clone Object 151](#_Toc152161421)

[Exercise 3: Merge Multiple Objects 155](#_Toc152161422)

[Object.entries 159](#_Toc152161423)

[**Exercise 1: Convert Object to Array of Key-Value Pairs** 159](#_Toc152161424)

[Exercise 2: Count Object Properties 163](#_Toc152161425)

[Exercise 3: Convert Object to Map 166](#_Toc152161426)

[Object.keys 171](#_Toc152161427)

[Exercise 1: Get All Property Names 171](#_Toc152161428)

[Exercise 2: Check Property Existence 175](#_Toc152161429)

[Exercise 3: Get Property Values 178](#_Toc152161430)

[Exercise 1: Calculate Age from Date of Birth 181](#_Toc152161431)

[Exercise 2: Days Until Next Birthday 182](#_Toc152161432)

[Exercise 1: Calculate Area of a Circle 184](#_Toc152161433)

[Exercise 2: Generate a Random Number within a Range 185](#_Toc152161434)

[Exercise 1: Format Number with Two Decimal Places 186](#_Toc152161435)

[Exercise 2: Check if a Value is an Integer 187](#_Toc152161436)

[find() 188](#_Toc152161437)

[Exercise 1: Find First Negative Number 188](#_Toc152161438)

[Exercise 2: Find First Capitalized Word 191](#_Toc152161439)

[Exercise 3: Find First Prime Number 193](#_Toc152161440)

[findIndex() 197](#_Toc152161441)

[Exercise 1: Find Index of First Negative Number 197](#_Toc152161442)

[Exercise 2: Find Index of First Capitalized Word 200](#_Toc152161443)

[Exercise 3: Find Index of First Prime Number 203](#_Toc152161444)

[sort() 206](#_Toc152161445)

[Exercise 1: Sort Numbers in Descending Order 206](#_Toc152161446)

[Exercise 2: Sort Words Alphabetically 209](#_Toc152161447)

[Exercise 3: Sort Strings by Length 212](#_Toc152161448)

[splice() 216](#_Toc152161449)

[Exercise 1: Remove Negative Numbers 216](#_Toc152161450)

[Exercise 2: Replace First Word 219](#_Toc152161451)

[Exercise 3: Insert Numbers 222](#_Toc152161452)

[slice() 225](#_Toc152161453)

[**Exercise 1: Extract Subarray** 225](#_Toc152161454)

[**Exercise 2: Extract Domain from Email** 228](#_Toc152161455)

[**Exercise 3: Extract Last 3 Characters** 231](#_Toc152161456)

[concat() 234](#_Toc152161457)

[**Exercise 1: Merge Two Arrays** 234](#_Toc152161458)

[**Exercise 2: Concatenate Words** 237](#_Toc152161459)

[**Exercise 3: Concatenate Multiple Arrays** 240](#_Toc152161460)

[join() 243](#_Toc152161461)

[**Exercise 1: Create a Sentence** 243](#_Toc152161462)

[**Exercise 2: Create a Comma-Separated List** 246](#_Toc152161463)

[**Exercise 3: Create a Dashed String** 249](#_Toc152161464)

# 

# 

# **Javascript Map method**

## **Exercise 1: Calculate Customer Score Using map()**

**Problem Statement**

Given an array of customers, each with a name, an array of purchases, and a location, calculate the "customer score" for each customer and add it as a new property customerScore to each customer object. The customer score is calculated based on the following formula:

*Customer Score=Total Amount Spent × Number of Purchases*

Here, Total Amount Spent is the sum of all the amounts in the purchases array for each customer, and Number of Purchases is the length of the purchases array for each customer.

**Sample Data**

|  |
| --- |
| const customers = [  { name: "Alice", purchases: [100, 50, 400], location: "USA" },  { name: "Bob", purchases: [200, 300], location: "Canada" },  { name: "Charlie", purchases: [300, 100, 150, 200], location: "UK" },  { name: "Dave", purchases: [50], location: "Australia" } ]; |

**Code Template**

|  |
| --- |
| // Complete this function function calculateCustomerScore(customers) {  // Use map() and reduce() to calculate customerScore for each customer }  // Test your function const updatedCustomers = calculateCustomerScore(customers); console.log(updatedCustomers); |

**Expected Output**

|  |
| --- |
| [  { name: "Alice", purchases: [100, 50, 400], location: "USA", customerScore: 1650 },  { name: "Bob", purchases: [200, 300], location: "Canada", customerScore: 1000 },  { name: "Charlie", purchases: [300, 100, 150, 200], location: "UK", customerScore: 3000 },  { name: "Dave", purchases: [50], location: "Australia", customerScore: 50 } ] |

**Solution:**

|  |
| --- |
| // Sample data: Array of customers with their names, purchases, and location const customers = [  { name: "Alice", purchases: [100, 50, 400], location: "USA" },  { name: "Bob", purchases: [200, 300], location: "Canada" },  { name: "Charlie", purchases: [300, 100, 150, 200], location: "UK" },  { name: "Dave", purchases: [50], location: "Australia" } ];  // Function to calculate customer score for each customer function calculateCustomerScore(customers) {  // Using map() to transform each customer object in the array  return customers.map(customer => {  // Calculating the total amount spent by each customer  // by summing up the values in their purchases array  const totalSpent = customer.purchases.reduce((total, current) => total + current, 0);   // Calculating the number of purchases for each customer  const numOfPurchases = customer.purchases.length;   // Calculating the customer score using the formula  // Customer Score = Total Amount Spent × Number of Purchases  const customerScore = totalSpent \* numOfPurchases;   // Returning the modified customer object with the new customerScore property  return { ...customer, customerScore };  }); }  // Testing the function const updatedCustomers = calculateCustomerScore(customers); console.log(updatedCustomers); |

**Alternative solution using loop:**

|  |
| --- |
| const customers = [  { name: 'Alice', purchases: [100, 50, 400], location: 'USA' },  { name: 'Bob', purchases: [200, 300], location: 'Canada' },  { name: 'Charlie', purchases: [300, 100, 150, 200], location: 'UK' },  { name: 'Dave', purchases: [50], location: 'Australia' }, ]  // Function to calculate customer score for each customer using a loop function calculateCustomerScore(customers) {  // Initialize an empty array to store the updated customers  let updatedCustomers = []   // Loop through each customer in the customers array  for (let i = 0; i < customers.length; i++) {  // Access the current customer  const customer = customers[i]   // Initialize a variable to store the total amount spent  let totalSpent = 0   // Loop through the customer's purchases to calculate the total amount spent  for (let j = 0; j < customer.purchases.length; j++) {  totalSpent += customer.purchases[j]  }   // Calculate the number of purchases for the customer  const numOfPurchases = customer.purchases.length   // Calculate the customer score  const customerScore = totalSpent \* numOfPurchases   // Add the customerScore property to the customer object  // and push the updated object to the updatedCustomers array  updatedCustomers.push({ ...customer, customerScore })  }   // Return the array of updated customers  return updatedCustomers }  // Testing the function const updatedCustomers = calculateCustomerScore(customers) console.log(updatedCustomers) |

## **Exercise 2: Normalize Product Prices Using map()**

**Problem Statement**

In an e-commerce platform, products can be listed by multiple sellers, and each seller may price the same product differently. You are tasked to normalize the prices of each product based on the following formula:



Each product in the product list has a name, a category, and an array of prices from different sellers.

Your task is to create a new array of products where each product has a new property called normalizedPrices, which contains the normalized prices for each seller.

**Sample Data**

|  |
| --- |
| const products = [  { name: "Laptop", category: "Electronics", prices: [900, 1000, 1100] },  { name: "Smartphone", category: "Electronics", prices: [400, 450, 500] },  { name: "Shirt", category: "Apparel", prices: [20, 25, 30] },  { name: "Pants", category: "Apparel", prices: [40, 50, 60] } ]; |

**Code Template**

|  |
| --- |
| // Complete this function function normalizeProductPrices(products) {  // Use map() method here to normalize the prices for each product }  // Test your function const normalizedProducts = normalizeProductPrices(products); console.log(normalizedProducts); |

**Expected Output**

|  |
| --- |
| [  { name: "Laptop", category: "Electronics", prices: [900, 1000, 1100], normalizedPrices: [0, 50, 100] },  { name: "Smartphone", category: "Electronics", prices: [400, 450, 500], normalizedPrices: [0, 50, 100] },  { name: "Shirt", category: "Apparel", prices: [20, 25, 30], normalizedPrices: [0, 50, 100] },  { name: "Pants", category: "Apparel", prices: [40, 50, 60], normalizedPrices: [0, 50, 100] } ] |

**Solution:**

|  |
| --- |
| // Sample data: Array of products with their names, categories, and prices const products = [  { name: "Laptop", category: "Electronics", prices: [900, 1000, 1100] },  { name: "Smartphone", category: "Electronics", prices: [400, 450, 500] },  { name: "Shirt", category: "Apparel", prices: [20, 25, 30] },  { name: "Pants", category: "Apparel", prices: [40, 50, 60] } ];  // Function to normalize product prices function normalizeProductPrices(products) {  // Using map() to transform each product object in the array  return products.map(product => {  // Find the minimum price for the product  const minPrice = Math.min(...product.prices);   // Calculate normalized prices relative to the minimum price  const normalizedPrices = product.prices.map(price => price - minPrice);  // Returning the modified product object with the new normalizedPrices property  return { ...product, normalizedPrices };  }); }  // Testing the function const normalizedProducts = normalizeProductPrices(products); console.log(normalizedProducts); |

**Alternative solution using loop:**

|  |
| --- |
| // Sample data: Array of products with their names, categories, and prices const products = [  { name: "Laptop", category: "Electronics", prices: [900, 1000, 1100] },  { name: "Smartphone", category: "Electronics", prices: [400, 450, 500] },  { name: "Shirt", category: "Apparel", prices: [20, 25, 30] },  { name: "Pants", category: "Apparel", prices: [40, 50, 60] } ];  // Function to normalize product prices using a loop function normalizeProductPrices(products) {  // Initialize an empty array to store the normalized products  let normalizedProducts = [];   // Loop through each product in the products array  for (let i = 0; i < products.length; i++) {  // Access the current product  const product = products[i];   // Find the minimum price for the product  let minPrice = Math.min(...product.prices);   // Initialize an array to store the normalized prices  let normalizedPrices = [];   // Loop through the product's prices to calculate normalized prices  for (let j = 0; j < product.prices.length; j++) {  normalizedPrices.push(product.prices[j] - minPrice);  }   // Add the normalizedPrices property to the product object  // and push the updated object to the normalizedProducts array  normalizedProducts.push({ ...product, normalizedPrices });  }   // Return the array of normalized products  return normalizedProducts; }  // Testing the function const normalizedProducts = normalizeProductPrices(products); console.log(normalizedProducts); |

## **Exercise 3: Convert Object Array to Array of Values**

**Problem Statement:**

Given an array of objects where each object represents a user with a name and age, convert this array into an array of names.

|  |
| --- |
| const users = [  { name: "John", age: 25 },  { name: "Jane", age: 30 },  { name: "Doe", age: 35 } ]; |

**Code Template:**

|  |
| --- |
| function extractNames(users) {  // Use map() to extract names from the users array } const names = extractNames(users); console.log(names); |

**Expected Output:**

|  |
| --- |
| ["John", "Jane", "Doe"] |

|  |
| --- |
| **Solution:**  // Sample data: Array of user objects with 'name' and 'age' properties const users = [  { name: "John", age: 25 },  { name: "Jane", age: 30 },  { name: "Doe", age: 35 } ];  // Function to extract names from the users array function extractNames(users) {  // Using map() to transform each user object into just the user's name  return users.map(user => user.name); }  // Testing the function const names = extractNames(users); console.log(names); // Output: ["John", "Jane", "Doe"]  Alternative solution using loop:  // Sample data: Array of user objects with 'name' and 'age' properties const users = [  { name: "John", age: 25 },  { name: "Jane", age: 30 },  { name: "Doe", age: 35 } ];  // Function to extract names from the users array using a loop function extractNames(users) {  // Initialize an empty array to store the names  let names = [];   // Loop through each user in the users array  for (let i = 0; i < users.length; i++) {  // Add the name of each user to the names array  names.push(users[i].name);  }   // Return the array of names  return names; }  // Testing the function const names = extractNames(users); console.log(names); // Output: ["John", "Jane", "Doe"] |

## 

## **Exercise 4: Calculate Total Price with Tax**

**Problem Statement:**

Given an array of product prices, calculate the total price for each product including a tax of 10%.

|  |
| --- |
| const prices = [100, 200, 300]; |

**Code Template:**

|  |
| --- |
| function calculateTotalWithTax(prices) {  // Use map() to calculate the total price with tax for each product } const pricesWithTax = calculateTotalWithTax(prices); console.log(pricesWithTax); |

**Expected Output:**

|  |
| --- |
| [110, 220, 330] |

## **Exercise 5: Convert Minutes to Hours and Minutes**

**Problem Statement:**

Given an array of durations in minutes, convert each duration into hours and minutes.

|  |
| --- |
| const durations = [120, 150, 90]; |

**Code Template:**

|  |
| --- |
| function convertToHoursAndMinutes(durations) {  // Use map() to convert durations into hours and minutes } const convertedDurations = convertToHoursAndMinutes(durations); console.log(convertedDurations); |

**Expected Output:**

|  |
| --- |
| function convertToHoursAndMinutes(durations) {  // Use map() to convert durations into hours and minutes } const convertedDurations = convertToHoursAndMinutes(durations); console.log(convertedDurations); |

## **Exercise 6: Capitalize First Letter of Each Word**

**Problem Statement:**

Given an array of words, capitalize the first letter of each word.

|  |
| --- |
| const words = ["apple", "banana", "cherry"]; |

**Code Template:**

|  |
| --- |
| function capitalizeWords(words) {  // Use map() to capitalize the first letter of each word } const capitalizedWords = capitalizeWords(words); console.log(capitalizedWords); |

**Expected Output:**

|  |
| --- |
| ["Apple", "Banana", "Cherry"]  **Solution**:  // Sample data: Array of words const words = ["apple", "banana", "cherry"];  // Function to capitalize the first letter of each word function capitalizeWords(words) {  // Using map() to transform each word  return words.map(word => {  // Capitalizing the first letter and concatenating it with the rest of the word  return word.charAt(0).toUpperCase() + word.slice(1);  }); }  // Testing the function const capitalizedWords = capitalizeWords(words); console.log(capitalizedWords); // Output: ["Apple", "Banana", "Cherry"]  **Alternative solution using loop:**  // Sample data: Array of words const words = ["apple", "banana", "cherry"];  // Function to capitalize the first letter of each word using a loop function capitalizeWords(words) {  // Initialize an empty array to store the capitalized words  let capitalizedWords = [];   // Loop through each word in the words array  for (let i = 0; i < words.length; i++) {  // Capitalize the first letter and concatenate it with the rest of the word  const capitalizedWord = words[i].charAt(0).toUpperCase() + words[i].slice(1);   // Add the capitalized word to the array  capitalizedWords.push(capitalizedWord);  }   // Return the array of capitalized words  return capitalizedWords; }  // Testing the function const capitalizedWords = capitalizeWords(words); console.log(capitalizedWords); // Output: ["Apple", "Banana", "Cherry"] |

## **Exercise 7: Convert RGB to Hex**

**Problem Statement:**

Given an array of RGB values, convert each RGB value to its hex representation.

|  |
| --- |
| const rgbValues = [  { r: 255, g: 0, b: 0 },  { r: 0, g: 255, b: 0 },  { r: 0, g: 0, b: 255 } ]; |

**Code Template:**

|  |
| --- |
| function convertToHex(rgbValues) {  // Use map() to convert each RGB value to hex } const hexValues = convertToHex(rgbValues); console.log(hexValues); |

**Expected Output:**

|  |
| --- |
| ["#ff0000", "#00ff00", "#0000ff"]  **Solution:**  // Sample data: Array of RGB values const rgbValues = [  { r: 255, g: 0, b: 0 },  { r: 0, g: 255, b: 0 },  { r: 0, g: 0, b: 255 } ];  // Function to convert a single RGB value to hex function rgbToHex(r, g, b) {  // Convert each RGB component to a two-digit hexadecimal string  const toHex = component => {  const hex = component.toString(16);  return hex.length == 1 ? '0' + hex : hex;  };   // Concatenate the converted values into a single hex color string  return `#${toHex(r)}${toHex(g)}${toHex(b)}`; }  // Function to convert each RGB value in the array to hex function convertToHex(rgbValues) {  // Using map() to transform each RGB object to its hex representation  return rgbValues.map(rgb => rgbToHex(rgb.r, rgb.g, rgb.b)); }  // Testing the function const hexValues = convertToHex(rgbValues); console.log(hexValues); // Output: ["#ff0000", "#00ff00", "#0000ff"]  **Alternative solution using loop:**  // Sample data: Array of RGB values const rgbValues = [  { r: 255, g: 0, b: 0 },  { r: 0, g: 255, b: 0 },  { r: 0, g: 0, b: 255 } ];  // Function to convert a single RGB value to hex function rgbToHex(r, g, b) {  // Convert each RGB component to a two-digit hexadecimal string  const toHex = component => {  const hex = component.toString(16);  return hex.length == 1 ? '0' + hex : hex;  };   // Concatenate the converted values into a single hex color string  return `#${toHex(r)}${toHex(g)}${toHex(b)}`; }  // Function to convert each RGB value in the array to hex using a loop function convertToHex(rgbValues) {  // Initialize an empty array to store the hex values  let hexValues = [];   // Loop through each RGB object in the array  for (let i = 0; i < rgbValues.length; i++) {  // Convert the RGB object to a hex string and add it to the hexValues array  hexValues.push(rgbToHex(rgbValues[i].r, rgbValues[i].g, rgbValues[i].b));  }   // Return the array of hex values  return hexValues; }  // Testing the function const hexValues = convertToHex(rgbValues); console.log(hexValues); // Output: ["#ff0000", "#00ff00", "#0000ff"] |

## **Exercise 8: Calculate Distance Between Coordinates**

**Problem Statement:**

Given an array of coordinates where each coordinate has x and y values, calculate the distance of each coordinate from the origin (0,0).

|  |
| --- |
| const coordinates = [  { x: 3, y: 4 },  { x: 6, y: 8 },  { x: 8, y: 15 } ];  **Formula to calculate distance** |

**Code Template:**

|  |
| --- |
| function calculateDistanceFromOrigin(coordinates) {  // Use map() to calculate the distance of each coordinate from the origin } const distances = calculateDistanceFromOrigin(coordinates); console.log(distances); |

**Expected Output:**

|  |
| --- |
| [5, 10, 17]  **Solution:**  // Sample data: Array of coordinates const coordinates = [  { x: 3, y: 4 },  { x: 6, y: 8 },  { x: 8, y: 15 } ];  // Function to calculate the distance from the origin for each coordinate function calculateDistanceFromOrigin(coordinates) {  // Using map() to calculate the distance for each coordinate  return coordinates.map(coord => {  // Applying the Pythagorean theorem to calculate the distance  return Math.sqrt(coord.x \*\* 2 + coord.y \*\* 2);  }); }  // Testing the function const distances = calculateDistanceFromOrigin(coordinates); console.log(distances); // Output: [5, 10, 17]  **Alternative solution using loop:**  // Sample data: Array of coordinates const coordinates = [  { x: 3, y: 4 },  { x: 6, y: 8 },  { x: 8, y: 15 } ];  // Function to calculate the distance from the origin for each coordinate using a loop function calculateDistanceFromOrigin(coordinates) {  // Initialize an empty array to store the distances  let distances = [];   // Loop through each coordinate in the coordinates array  for (let i = 0; i < coordinates.length; i++) {  // Calculate the distance using the Pythagorean theorem  const distance = Math.sqrt(coordinates[i].x \*\* 2 + coordinates[i].y \*\* 2);   // Add the calculated distance to the distances array  distances.push(distance);  }   // Return the array of distances  return distances; }  // Testing the function const distances = calculateDistanceFromOrigin(coordinates); console.log(distances); // Output: [5, 10, 17] |

# **Javascript Reduce method**

## **Exercise 1: Calculate the total inventory value for all products using reduce().**

**Problem Statement**

Given an array of products, each with a name, unitsInStock, and unitPrice, calculate the total inventory value for all products.

The inventory value for each product is defined as:

*Inventory Value = Units in Stock × Unit Price*

**Sample Data**

|  |
| --- |
| const products = [  { name: "Laptop", unitsInStock: 5, unitPrice: 1000 },  { name: "Smartphone", unitsInStock: 10, unitPrice: 500 },  { name: "TV", unitsInStock: 3, unitPrice: 1500 },  { name: "Headphones", unitsInStock: 15, unitPrice: 100 } ]; |

**Code Template**

|  |
| --- |
| // Complete this function function calculateTotalInventoryValue(products) {  // Use reduce() to calculate the total inventory value }  // Test your function const totalInventoryValue = calculateTotalInventoryValue(products); console.log(`Total Inventory Value: $${totalInventoryValue}`); |

**Expected Output**

Your function should return the total inventory value:

*Total Inventory Value: $16000*

**Solution:**

// Sample data: Array of products  
const products = [  
 { name: "Laptop", unitsInStock: 5, unitPrice: 1000 },  
 { name: "Smartphone", unitsInStock: 10, unitPrice: 500 },  
 { name: "TV", unitsInStock: 3, unitPrice: 1500 },  
 { name: "Headphones", unitsInStock: 15, unitPrice: 100 }  
];  
  
// Function to calculate the total inventory value  
function calculateTotalInventoryValue(products) {  
 // Using reduce() to accumulate the total inventory value  
 return products.reduce((total, product) => {  
 // Calculating the inventory value for each product and adding it to the total  
 return total + (product.unitsInStock \* product.unitPrice);  
 }, 0); // Starting the accumulation from 0  
}  
  
// Testing the function  
const totalInventoryValue = calculateTotalInventoryValue(products);  
console.log(`Total Inventory Value: $${totalInventoryValue}`); // Output: Total Inventory Value: $16000

**Alternative solution with loop:**

// Sample data: Array of products  
const products = [  
 { name: "Laptop", unitsInStock: 5, unitPrice: 1000 },  
 { name: "Smartphone", unitsInStock: 10, unitPrice: 500 },  
 { name: "TV", unitsInStock: 3, unitPrice: 1500 },  
 { name: "Headphones", unitsInStock: 15, unitPrice: 100 }  
];  
  
// Function to calculate the total inventory value using a loop  
function calculateTotalInventoryValue(products) {  
 // Initialize a variable to store the total inventory value  
 let totalInventoryValue = 0;  
  
 // Loop through each product in the products array  
 for (let i = 0; i < products.length; i++) {  
 // Calculate the inventory value for each product and add it to the total  
 totalInventoryValue += products[i].unitsInStock \* products[i].unitPrice;  
 }  
  
 // Return the total inventory value  
 return totalInventoryValue;  
}  
  
// Testing the function  
const totalInventoryValue = calculateTotalInventoryValue(products);  
console.log(`Total Inventory Value: $${totalInventoryValue}`); // Output: Total Inventory Value: $16000

## **Exercise 2: Calculate Overall Customer Satisfaction Score Using reduce().**

**Problem Statement**

Given an array of customer ratings, calculate the overall customer satisfaction score. The score is defined as the average rating rounded to the nearest integer.

**Sample Data**

|  |
| --- |
| const customerRatings = [4.5, 3.8, 5.0, 4.2, 4.9]; |

**Code Template**

|  |
| --- |
| // Complete this function function calculateCustomerSatisfactionScore(ratings) {  // Use reduce() to calculate the overall customer satisfaction score }  // Test your function const overallScore = calculateCustomerSatisfactionScore(customerRatings); console.log(`Overall Customer Satisfaction Score: ${overallScore}`); |

**Expected Output**

Your function should return the overall customer satisfaction score:

Overall Customer Satisfaction Score: 4

**Solution:**

// Sample data: Array of customer ratings  
const customerRatings = [4.5, 3.8, 5.0, 4.2, 4.9];  
  
// Function to calculate the overall customer satisfaction score  
function calculateCustomerSatisfactionScore(ratings) {  
 // Using reduce() to sum up all the ratings  
 const total = ratings.reduce((sum, rating) => sum + rating, 0);  
  
 // Calculating the average rating  
 const average = total / ratings.length;  
  
 // Rounding the average to the nearest integer  
 return Math.round(average);  
}  
  
// Testing the function  
const overallScore = calculateCustomerSatisfactionScore(customerRatings);  
console.log(`Overall Customer Satisfaction Score: ${overallScore}`);

**Alternative solution using loop:**

// Sample data: Array of customer ratings  
const customerRatings = [4.5, 3.8, 5.0, 4.2, 4.9];  
  
// Function to calculate the overall customer satisfaction score using a loop  
function calculateCustomerSatisfactionScore(ratings) {  
 // Initialize variables for sum and count  
 let sum = 0;  
 let count = 0;  
  
 // Loop through each rating in the ratings array  
 for (let i = 0; i < ratings.length; i++) {  
 // Add the current rating to the sum  
 sum += ratings[i];  
  
 // Increment the count  
 count++;  
 }  
  
 // Calculate the average rating  
 const average = sum / count;  
  
 // Round the average to the nearest integer  
 return Math.round(average);  
}  
  
// Testing the function  
const overallScore = calculateCustomerSatisfactionScore(customerRatings);  
console.log(`Overall Customer Satisfaction Score: ${overallScore}`); // Output: 4

## **Exercise 3: Calculate Average Rating for Each Category**

**Problem Statement**

Given an array of products, each with a name, category, and rating, calculate the average rating for each product category.

The rating for each product is between 1 and 5.

**Sample Data**

|  |
| --- |
| const products = [  { name: "Laptop", category: "Electronics", rating: 4 },  { name: "Smartphone", category: "Electronics", rating: 5 },  { name: "TV", category: "Electronics", rating: 3 },  { name: "Shirt", category: "Apparel", rating: 5 },  { name: "Pants", category: "Apparel", rating: 4 },  { name: "Hat", category: "Apparel", rating: 3 } ]; |

**Code Template**

|  |
| --- |
| // Complete this function function averageRatingByCategory(products) {  // Use reduce() to calculate the average rating for each category }  // Test your function const averageRatings = averageRatingByCategory(products); console.log("Average Ratings by Category:", averageRatings); |

**Expected Output**

Your function should return an object that contains the average rating for each category:

|  |
| --- |
| Average Ratings by Category: { Electronics: 4, Apparel: 4 } |

**Solution**:

// Sample data with products, their categories, and ratings  
const products = [  
 { name: "Laptop", category: "Electronics", rating: 4 },  
 { name: "Smartphone", category: "Electronics", rating: 5 },  
 { name: "TV", category: "Electronics", rating: 3 },  
 { name: "Shirt", category: "Apparel", rating: 5 },  
 { name: "Pants", category: "Apparel", rating: 4 },  
 { name: "Hat", category: "Apparel", rating: 3 }  
];  
  
// Function to calculate the average rating for each category  
function averageRatingByCategory(products) {  
 // We use reduce() to build up an intermediate object that will help in calculating average  
 const intermediateResult = products.reduce((acc, product) => {  
 // If category is not yet a key in our accumulator, initialize it  
 if (!acc[product.category]) {  
 acc[product.category] = { totalRating: 0, count: 0 };  
 }  
   
 // Update total rating and count for the category  
 acc[product.category].totalRating += product.rating;  
 acc[product.category].count += 1;  
  
 return acc;  
 }, {});  
  
 // Create a new object to store average ratings  
 const averageRatings = {};  
  
 // Calculate average for each category and assign to the new object  
 for (let category in intermediateResult) {  
 averageRatings[category] = intermediateResult[category].totalRating / intermediateResult[category].count;  
 }  
  
 return averageRatings;  
}  
  
// Test the function and print the results  
const averageRatings = averageRatingByCategory(products);  
console.log("Average Ratings by Category:", averageRatings); // Expected: { Electronics: 4, Apparel: 4 }

**Alternative solution using loop:**

// Sample data with products, their categories, and ratings  
const products = [  
 { name: 'Laptop', category: 'Electronics', rating: 4 },  
 { name: 'Smartphone', category: 'Electronics', rating: 5 },  
 { name: 'TV', category: 'Electronics', rating: 3 },  
 { name: 'Shirt', category: 'Apparel', rating: 5 },  
 { name: 'Pants', category: 'Apparel', rating: 4 },  
 { name: 'Hat', category: 'Apparel', rating: 3 },  
]  
  
// Function to calculate the average rating for each category using loops  
function averageRatingByCategory(products) {  
 // Initialize an object to store intermediate results: total ratings and count for each category  
 const intermediateResult = {}  
  
 // Iterate over each product in the products array  
 for (let i = 0; i < products.length; i++) {  
 const product = products[i]  
  
 // If the category of the current product doesn't exist in intermediateResult, initialize it  
 if (!intermediateResult[product.category]) {  
 intermediateResult[product.category] = { totalRating: 0, count: 0 }  
 }  
  
 // Update the total rating and count for the product's category  
 intermediateResult[product.category].totalRating += product.rating  
 intermediateResult[product.category].count += 1  
 }  
  
 // Initialize an object to store the final average ratings for each category  
 const averageRatings = {}  
  
 // Iterate over each category in intermediateResult to calculate average ratings  
 for (let category in intermediateResult) {  
 averageRatings[category] =  
 intermediateResult[category].totalRating /  
 intermediateResult[category].count  
 }  
  
 return averageRatings  
}  
  
// Test the function and print the results  
const averageRatings = averageRatingByCategory(products)  
console.log('Average Ratings by Category:', averageRatings) // Expected: { Electronics: 4, Apparel: 4 }

## **Exercise 4: Find Most Frequent Words in Text**

**Problem Statement:**

Given a text string, find the frequency of each word in the text.

**Sample Data:**

|  |
| --- |
| const text = "apple banana apple orange banana apple"; |

**Code Template**:

|  |
| --- |
| // Import necessary dependencies if you're using a data analytics tool  // Complete this function function findMostFrequentWords(text) {  // Use reduce() to find frequency of each word }  // Test your function const wordFrequencies = findMostFrequentWords(text); console.log("Word Frequencies:", wordFrequencies); |

**Expected Output:**

Your function should return an object that contains the frequency of each word:

|  |
| --- |
| Word Frequencies: { apple: 3, banana: 2, orange: 1 } |

**Solution:**

// Sample data: Text string  
const text = "apple banana apple orange banana apple";  
  
// Function to find the frequency of each word in the text  
function findMostFrequentWords(text) {  
 // Split the text into words  
 const words = text.split(" ");  
  
 // Using reduce() to calculate the frequency of each word  
 return words.reduce((frequencies, word) => {  
 // If the word is already in the object, increment its count  
 // Otherwise, set its count to 1  
 frequencies[word] = (frequencies[word] || 0) + 1;  
 return frequencies;  
 }, {}); // Start with an empty object  
}  
  
// Testing the function  
const wordFrequencies = findMostFrequentWords(text);  
console.log("Word Frequencies:", wordFrequencies);

**Alternative solution using loop:**

// Sample data: Text string  
const text = "apple banana apple orange banana apple";  
  
// Function to find the frequency of each word in the text using a loop  
function findMostFrequentWords(text) {  
 // Split the text into words  
 const words = text.split(" ");  
  
 // Initialize an object to store the word frequencies  
 let frequencies = {};  
  
 // Loop through each word in the words array  
 for (let i = 0; i < words.length; i++) {  
 // If the word is already in the frequencies object, increment its count  
 if (frequencies[words[i]]) {  
 frequencies[words[i]] += 1;  
 } else {  
 // Otherwise, set its count to 1  
 frequencies[words[i]] = 1;  
 }  
 }  
  
 // Return the object containing the word frequencies  
 return frequencies;  
}  
  
// Testing the function  
const wordFrequencies = findMostFrequentWords(text);  
console.log("Word Frequencies:", wordFrequencies); // Output: { apple: 3, banana: 2, orange: 1 }

## **Exercise 5. Calculate Cumulative Monthly Sales**

**Problem Statement:**

Given an array of sales records with month and revenue data, calculate the cumulative sales for each month.

**Sample Data:**

|  |
| --- |
| const sales = [  { month: "January", revenue: 1000 },  { month: "February", revenue: 1500 },  { month: "March", revenue: 1200 } ]; |

**Code Template:**

|  |
| --- |
| function calculateCumulativeSales(sales) {  // Use reduce() to calculate cumulative sales for each month }  // Test your function const cumulativeSales = calculateCumulativeSales(sales); console.log("Cumulative Monthly Sales:", cumulativeSales); |

**Expected Output:**

|  |
| --- |
| Cumulative Monthly Sales: { January: 1000, February: 2500, March: 3700 }  **Solution:**  // Sample data: Array of sales records const sales = [  { month: "January", revenue: 1000 },  { month: "February", revenue: 1500 },  { month: "March", revenue: 1200 } ];  // Function to calculate cumulative sales for each month function calculateCumulativeSales(sales) {  // Using reduce() to calculate cumulative sales  return sales.reduce((cumulative, record, index) => {  // Add the current month's revenue to the total  // If it's the first record, the total is just the current revenue  const total = index === 0 ? record.revenue : cumulative[sales[index - 1].month] + record.revenue;   // Add the cumulative total for the current month to the result object  cumulative[record.month] = total;  return cumulative;  }, {}); // Start with an empty object }  // Testing the function const cumulativeSales = calculateCumulativeSales(sales); console.log("Cumulative Monthly Sales:", cumulativeSales);  **Alternative solution using loop:**  // Sample data: Array of sales records const sales = [  { month: "January", revenue: 1000 },  { month: "February", revenue: 1500 },  { month: "March", revenue: 1200 } ];  // Function to calculate cumulative sales for each month using a loop function calculateCumulativeSales(sales) {  // Initialize an object to store the cumulative sales  let cumulativeSales = {};  let cumulativeTotal = 0;   // Loop through each sale record in the sales array  for (let i = 0; i < sales.length; i++) {  // Add the current month's revenue to the cumulative total  cumulativeTotal += sales[i].revenue;   // Store the cumulative total in the corresponding month key  cumulativeSales[sales[i].month] = cumulativeTotal;  }   // Return the object containing the cumulative sales  return cumulativeSales; }  // Testing the function const cumulativeSales = calculateCumulativeSales(sales); console.log("Cumulative Monthly Sales:", cumulativeSales); // Output: { January: 1000, February: 2500, March: 3700 } |

## **Exercise 6. Group Data by Multiple Criteria**

**Problem Statement:**

Given an array of products with category and brand data, group the products by both category and brand.

**Sample Data:**

|  |
| --- |
| const products = [  { name: "A", category: "Electronics", brand: "Sony" },  { name: "B", category: "Electronics", brand: "Apple" },  { name: "C", category: "Apparel", brand: "Nike" },  { name: "D", category: "Electronics", brand: "Sony" } ]; |

**Code Template:**

|  |
| --- |
| function groupByCategoryAndBrand(products) {  // Use reduce() to group products by both category and brand }  // Test your function const groupedProducts = groupByCategoryAndBrand(products); console.log("Products Grouped by Category and Brand:", groupedProducts); |

**Expected Output:**

|  |
| --- |
| Products Grouped by Category and Brand:  {  Electronics: { Sony: ["A", "D"], Apple: ["B"] },  Apparel: { Nike: ["C"] } } |

**Solution**:

// Sample data: Array of products with category and brand  
const products = [  
 { name: "A", category: "Electronics", brand: "Sony" },  
 { name: "B", category: "Electronics", brand: "Apple" },  
 { name: "C", category: "Apparel", brand: "Nike" },  
 { name: "D", category: "Electronics", brand: "Sony" }  
];  
  
// Function to group products by both category and brand  
function groupByCategoryAndBrand(products) {  
 // Using reduce() to group products  
 return products.reduce((grouped, product) => {  
 // If the category doesn't exist yet, create it  
 if (!grouped[product.category]) {  
 grouped[product.category] = {};  
 }  
 // If the brand within the category doesn't exist, create it  
 if (!grouped[product.category][product.brand]) {  
 grouped[product.category][product.brand] = [];  
 }  
 // Add the product name to the corresponding brand array  
 grouped[product.category][product.brand].push(product.name);  
 return grouped;  
 }, {}); // Start with an empty object  
}  
  
// Testing the function  
const groupedProducts = groupByCategoryAndBrand(products);  
console.log("Products Grouped by Category and Brand:", groupedProducts);

**Alternative solution using loop:**

// Sample data: Array of products with category and brand  
const products = [  
 { name: "A", category: "Electronics", brand: "Sony" },  
 { name: "B", category: "Electronics", brand: "Apple" },  
 { name: "C", category: "Apparel", brand: "Nike" },  
 { name: "D", category: "Electronics", brand: "Sony" }  
];  
  
// Function to group products by both category and brand using loops  
function groupByCategoryAndBrand(products) {  
 // Initialize an object to store the grouped products  
 let groupedProducts = {};  
  
 // Loop through each product in the products array  
 for (let i = 0; i < products.length; i++) {  
 let product = products[i];  
 let category = product.category;  
 let brand = product.brand;  
  
 // If the category doesn't exist yet, create it  
 if (!groupedProducts[category]) {  
 groupedProducts[category] = {};  
 }  
 // If the brand within the category doesn't exist, create it  
 if (!groupedProducts[category][brand]) {  
 groupedProducts[category][brand] = [];  
 }  
 // Add the product name to the corresponding brand array  
 groupedProducts[category][brand].push(product.name);  
 }  
  
 // Return the object containing the grouped products  
 return groupedProducts;  
}  
  
// Testing the function  
const groupedProducts = groupByCategoryAndBrand(products);  
console.log("Products Grouped by Category and Brand:", groupedProducts);

## Exercise 7. Compute Weighted Average

**Problem Statement:**

Given an array of grades with scores and their respective weights, calculate the weighted average.

**Sample Data:**

|  |
| --- |
| const grades = [  { score: 90, weight: 0.4 },  { score: 85, weight: 0.6 } ];  Formulae **to calculate Weighted Average:** |

**Code Template:**

|  |
| --- |
| function calculateWeightedAverage(grades) {  // Use reduce() to compute the weighted average }  // Test your function const weightedAvg = calculateWeightedAverage(grades); console.log("Weighted Average:", weightedAvg); |

**Expected Output:**

|  |
| --- |
| *Weighted Average: 87* |

Solution:

// Sample data: Array of grades with scores and weights

const grades = [

{ score: 90, weight: 0.4 },

{ score: 85, weight: 0.6 }

];

// Function to calculate the weighted average

function calculateWeightedAverage(grades) {

// Sum up the products of scores and weights using reduce()

const totalScore = grades.reduce((sum, grade) => sum + grade.score \* grade.weight, 0);

// Sum up the total weight

const totalWeight = grades.reduce((sum, grade) => sum + grade.weight, 0);

// Calculate the weighted average

return totalScore / totalWeight;

}

// Testing the function

const weightedAvg = calculateWeightedAverage(grades);

console.log("Weighted Average:", weightedAvg.toFixed(0)); // Output: Weighted Average: 87

Alternative solution using loop:

// Sample data: Array of grades with scores and weights

const grades = [

{ score: 90, weight: 0.4 },

{ score: 85, weight: 0.6 }

];

// Function to calculate the weighted average using loops

function calculateWeightedAverage(grades) {

// Initialize variables for summing scores and weights

let totalScore = 0;

let totalWeight = 0;

// Loop through each grade in the grades array

for (let i = 0; i < grades.length; i++) {

// Multiply each score by its weight and add it to the total score

totalScore += grades[i].score \* grades[i].weight;

// Add each weight to the total weight

totalWeight += grades[i].weight;

}

// Calculate the weighted average

return totalScore / totalWeight;

}

// Testing the function

const weightedAvg = calculateWeightedAverage(grades);

console.log("Weighted Average:", weightedAvg.toFixed(0)); // Output: Weighted Average: 87

## Exercise 8. Flatten Nested Data Structures

**Problem Statement:**

Given a nested array of numbers, flatten the array to a single-level array.

**Sample Data:**

|  |
| --- |
| const nestedArray = [1, [2, 3], [4, [5, 6]]]; |

**Code Template:**

|  |
| --- |
| function flattenArray(nestedArray) {  // Use reduce() to flatten the nested array }  // Test your function const flattened = flattenArray(nestedArray); console.log("Flattened Array:", flattened); |

**Expected Output:**

*Flattened Array: [1, 2, 3, 4, 5, 6]*

Solution:

// Sample data: Nested array of numbers

const nestedArray = [1, [2, 3], [4, [5, 6]]];

// Function to flatten a nested array

function flattenArray(nestedArray) {

// Using reduce() to iterate and flatten the array

return nestedArray.reduce((flattened, element) => {

// If the element is an array, recursively flatten it and concatenate

// Otherwise, just concatenate the element

return Array.isArray(element)

? flattened.concat(flattenArray(element))

: flattened.concat(element);

}, []); // Start with an empty array

}

// Testing the function

const flattened = flattenArray(nestedArray);

console.log("Flattened Array:", flattened); // Output: Flattened Array: [1, 2, 3, 4, 5, 6]

Alternative solution using loop:

// Sample data: Nested array of numbers

const nestedArray = [1, [2, 3], [4, [5, 6]]];

// Recursive function to flatten an array

function flattenArray(arr) {

let result = [];

// Loop through each element in the array

for (let i = 0; i < arr.length; i++) {

if (Array.isArray(arr[i])) {

// If the element is an array, recursively flatten it and add the results to the result array

result = result.concat(flattenArray(arr[i]));

} else {

// If the element is not an array, add it directly to the result array

result.push(arr[i]);

}

}

return result;

}

// Testing the function

const flattened = flattenArray(nestedArray);

console.log("Flattened Array:", flattened); // Output: Flattened Array: [1, 2, 3, 4, 5, 6]

# filter()

## Exercise 1: Filter Out Inactive Users

**Problem Statement:**

Given an array of users, each with a name and an isActive boolean property, filter out all the inactive users.

|  |
| --- |
| const users = [  { name: "John", isActive: true },  { name: "Jane", isActive: false },  { name: "Doe", isActive: true } ]; |

**Code Template:**

|  |
| --- |
| function getActiveUsers(users) {  // Use filter() to get all active users } const activeUsers = getActiveUsers(users); console.log(activeUsers); |

**Expected Output:**

|  |
| --- |
| [  { name: "John", isActive: true },  { name: "Doe", isActive: true } ] |

Solution:

// Given array of users with 'name' and 'isActive' properties

const users = [

{ name: "John", isActive: true },

{ name: "Jane", isActive: false },

{ name: "Doe", isActive: true }

];

function getActiveUsers(users) {

// The filter() method creates a new array with all elements

// that pass the test implemented by the provided function.

// Here, we're using an arrow function to check the 'isActive'

// property of each user. If 'isActive' is true, the user object

// will be included in the new array, otherwise, it will be omitted.

return users.filter(user => user.isActive);

}

// Calling the function and storing the result in the 'activeUsers' constant

const activeUsers = getActiveUsers(users);

// Printing the result to the console

console.log(activeUsers);

Alternative solution using loop:

// Given array of users with 'name' and 'isActive' properties

const users = [

{ name: "John", isActive: true },

{ name: "Jane", isActive: false },

{ name: "Doe", isActive: true }

];

function getActiveUsers(users) {

// Initialize an empty array to store the active users

let activeUsers = [];

// Use a for loop to iterate over each user in the users array

for (let i = 0; i < users.length; i++) {

// If the user's 'isActive' property is true,

// push the user object to the 'activeUsers' array

if (users[i].isActive) {

activeUsers.push(users[i]);

}

}

// Return the filtered 'activeUsers' array

return activeUsers;

}

// Calling the function and storing the result in the 'resultingActiveUsers' constant

const resultingActiveUsers = getActiveUsers(users);

// Printing the result to the console

console.log(resultingActiveUsers);

## Exercise 2: Filter Products Above a Price Range

**Problem Statement:**

Given an array of products, each with a name and a price, filter out all products that are priced below or equal to $100.

|  |
| --- |
| const products = [  { name: "Laptop", price: 150 },  { name: "Mouse", price: 25 },  { name: "Keyboard", price: 50 } ]; |

**Code Template:**

|  |
| --- |
| function getAffordableProducts(products) {  // Use filter() to get products priced below or equal to $100 } const affordableProducts = getAffordableProducts(products); console.log(affordableProducts); |

**Expected Output:**

|  |
| --- |
| [  { name: "Mouse", price: 25 },  { name: "Keyboard", price: 50 } ] |

Solution:

// Define an array of products with their names and prices.

const products = [

{ name: 'Laptop', price: 150 },

{ name: 'Mouse', price: 25 },

{ name: 'Keyboard', price: 50 },

]

// Define a function to filter products based on their price.

function getAffordableProducts(products) {

// Use the filter() method on the products array to filter products.

// The filter() method creates a new array with all elements that pass the test provided by the callback function.

return products.filter((product) => {

// Check if the product's price is less than or equal to 100.

// If true, the product will be included in the resulting array.

return product.price <= 100

})

}

// Call the getAffordableProducts function and store the result in the affordableProducts variable.

const affordableProducts = getAffordableProducts(products)

// Log the filtered products to the console.

console.log(affordableProducts)

Alternative solution:

// Define an array of products with their names and prices.

const products = [

{ name: "Laptop", price: 150 },

{ name: "Mouse", price: 25 },

{ name: "Keyboard", price: 50 }

];

// Define a function to filter products based on their price using a loop.

function getAffordableProducts(products) {

// Initialize an empty array to store the affordable products.

let affordableProducts = [];

// Use a for loop to iterate over each product in the products array.

for (let i = 0; i < products.length; i++) {

// Check if the current product's price is less than or equal to 100.

if (products[i].price <= 100) {

// If the condition is met, add the current product to the affordableProducts array.

affordableProducts.push(products[i]);

}

}

// Return the filtered list of affordable products.

return affordableProducts;

}

// Call the getAffordableProducts function and store the result in the affordableProducts variable.

const affordableProductsList = getAffordableProducts(products);

// Log the filtered products to the console.

console.log(affordableProductsList);

## Exercise 3: Filter Out Unpublished Articles

**Problem Statement:**

Given an array of articles, each with a title, content, and a isPublished boolean property, filter out all the unpublished articles.

|  |
| --- |
| const articles = [  { title: "JS Basics", content: "Lorem ipsum...", isPublished: true },  { title: "Advanced React", content: "Lorem ipsum...", isPublished: false },  { title: "Vue vs React", content: "Lorem ipsum...", isPublished: true } ]; |

**Code Template:**

|  |
| --- |
| function getPublishedArticles(articles) {  // Use filter() to get all published articles } const publishedArticles = getPublishedArticles(articles); console.log(publishedArticles); |

**Expected Output**:

|  |
| --- |
| [  { title: "JS Basics", content: "Lorem ipsum...", isPublished: true },  { title: "Vue vs React", content: "Lorem ipsum...", isPublished: true } ]  Solution:  // Sample data: Array of articles  const articles = [  { title: "JS Basics", content: "Lorem ipsum...", isPublished: true },  { title: "Advanced React", content: "Lorem ipsum...", isPublished: false },  { title: "Vue vs React", content: "Lorem ipsum...", isPublished: true }  ];  // Function to get all published articles  function getPublishedArticles(articles) {  // Using filter() to return only those articles where isPublished is true  return articles.filter(article => article.isPublished);  }  // Testing the function  const publishedArticles = getPublishedArticles(articles);  console.log(publishedArticles);  Alternative solution using loop:  // Sample data: Array of articles  const articles = [  { title: "JS Basics", content: "Lorem ipsum...", isPublished: true },  { title: "Advanced React", content: "Lorem ipsum...", isPublished: false },  { title: "Vue vs React", content: "Lorem ipsum...", isPublished: true }  ];  // Function to get all published articles using a loop  function getPublishedArticles(articles) {  // Initialize an array to hold published articles  let publishedArticles = [];  // Loop through each article in the articles array  for (let i = 0; i < articles.length; i++) {  // If the article is published, add it to the publishedArticles array  if (articles[i].isPublished) {  publishedArticles.push(articles[i]);  }  }  // Return the array of published articles  return publishedArticles;  }  // Testing the function  const publishedArticles = getPublishedArticles(articles);  console.log(publishedArticles); |

## Exercise 4: Filter Out Users Not in a Given Country

**Problem Statement**:

Given an array of users, each with a name and a country, filter out all users who are not from "USA".

|  |
| --- |
| const users = [  { name: "John", country: "USA" },  { name: "Jane", country: "Canada" },  { name: "Doe", country: "USA" } ]; |

**Code Template**:

function getUsersFromUSA(users) {

// Use filter() to get all users from "USA"

}

const usaUsers = getUsersFromUSA(users);

console.log(usaUsers);

**Expected Output**:

|  |
| --- |
| [  { name: "John", country: "USA" },  { name: "Doe", country: "USA" } ]  Solution:  // Sample data: Array of users  const users = [  { name: "John", country: "USA" },  { name: "Jane", country: "Canada" },  { name: "Doe", country: "USA" }  ];  // Function to get all users from "USA"  function getUsersFromUSA(users) {  // Using filter() to return only those users whose country is "USA"  return users.filter(user => user.country === "USA");  }  // Testing the function  const usaUsers = getUsersFromUSA(users);  console.log(usaUsers);  Alternative solution using loop:  // Sample data: Array of users  const users = [  { name: "John", country: "USA" },  { name: "Jane", country: "Canada" },  { name: "Doe", country: "USA" }  ];  // Function to get all users from "USA" using a loop  function getUsersFromUSA(users) {  // Initialize an array to hold users from the USA  let usaUsers = [];  // Loop through each user in the users array  for (let i = 0; i < users.length; i++) {  // If the user's country is "USA", add them to the usaUsers array  if (users[i].country === "USA") {  usaUsers.push(users[i]);  }  }  // Return the array of users from the USA  return usaUsers;  }  // Testing the function  const usaUsers = getUsersFromUSA(users);  console.log(usaUsers); |

# forEach()

## Exercise 1: Print User Emails

**Problem Statement**:

Given an array of users, each with a name and an email, print out each user's email.

|  |
| --- |
| const users = [  { name: "John", email: "john@example.com" },  { name: "Jane", email: "jane@example.com" },  { name: "Doe", email: "doe@example.com" } ]; |

**Code Template**:

|  |
| --- |
| function printUserEmails(users) {  // Use forEach() to print each user's email } printUserEmails(users); |

**Expected Output**:

|  |
| --- |
| john@example.com jane@example.com doe@example.com  Solution:  // Define an array of users with name and email properties  const users = [  { name: "John", email: "john@example.com" },  { name: "Jane", email: "jane@example.com" },  { name: "Doe", email: "doe@example.com" }  ];  // Define the function printUserEmails to print out emails of the users  function printUserEmails(users) {  // The forEach() method executes a provided function once for each array element.  // Here, we're using forEach to iterate over each user in the users array.  users.forEach(function(user) {  // For each user, we simply print (or log) their email to the console.  console.log(user.email);  });  }  // Call the printUserEmails function with the users array as its argument  printUserEmails(users);  // This would print:  // john@example.com  // jane@example.com  // doe@example.com  Alternative solution: |

// Define an array of users with name and email properties

const users = [

{ name: "John", email: "john@example.com" },

{ name: "Jane", email: "jane@example.com" },

{ name: "Doe", email: "doe@example.com" }

];

// Define the function printUserEmails to print out emails of the users

function printUserEmails(users) {

// Traditional for loop to iterate over the length of the users array

for (let i = 0; i < users.length; i++) {

// Access the email of the user at the current index (i) and log it to the console

console.log(users[i].email);

}

}

// Call the printUserEmails function with the users array as its argument

printUserEmails(users);

// This would print:

// john@example.com

// jane@example.com

// doe@example.com

## Exercise 2: Update Stock Quantity

**Problem Statement**:

Given an array of products, each with a name and a stockQuantity, decrement the stock quantity of each product by 1.

|  |
| --- |
| const products = [  { name: "Laptop", stockQuantity: 5 },  { name: "Mouse", stockQuantity: 10 },  { name: "Keyboard", stockQuantity: 7 } ]; |

**Code Template**:

|  |
| --- |
| function updateStock(products) {  // Use forEach() to decrement the stock quantity of each product by 1 } updateStock(products); console.log(products); |

**Expected Output**:

|  |
| --- |
| [  { name: "Laptop", stockQuantity: 4 },  { name: "Mouse", stockQuantity: 9 },  { name: "Keyboard", stockQuantity: 6 } ]  Solution:  // Sample data: Array of products  const products = [  { name: "Laptop", stockQuantity: 5 },  { name: "Mouse", stockQuantity: 10 },  { name: "Keyboard", stockQuantity: 7 }  ];  // Function to update the stock of each product  function updateStock(products) {  // Using forEach() to decrement the stock quantity of each product by 1  products.forEach(product => {  product.stockQuantity -= 1;  });  }  // Updating the stock  updateStock(products);  console.log(products);  Alternative solution using loop:  // Sample data: Array of products  const products = [  { name: "Laptop", stockQuantity: 5 },  { name: "Mouse", stockQuantity: 10 },  { name: "Keyboard", stockQuantity: 7 }  ];  // Function to update the stock of each product using a loop  function updateStock(products) {  // Loop through each product in the products array  for (let i = 0; i < products.length; i++) {  // Decrement the stock quantity of each product by 1  products[i].stockQuantity -= 1;  }  }  // Updating the stock  updateStock(products);  console.log(products); |

## Exercise 3: Send Notification to Users

**Problem Statement**:

Given an array of users, each with a name and a hasUnreadNotification boolean property, send a notification to each user who has an unread notification.

|  |
| --- |
| const users = [  { name: "John", hasUnreadNotification: true },  { name: "Jane", hasUnreadNotification: false },  { name: "Doe", hasUnreadNotification: true } ]; |

**Code Template**:

|  |
| --- |
| function sendNotifications(users) {  // Use forEach() to send a notification to each user with an unread notification } sendNotifications(users); |

**Expected Output**:

|  |
| --- |
| Notification sent to John Notification sent to Doe  Solution:  // Sample data: Array of users  const users = [  { name: "John", hasUnreadNotification: true },  { name: "Jane", hasUnreadNotification: false },  { name: "Doe", hasUnreadNotification: true }  ];  // Function to send notifications to users with unread notifications  function sendNotifications(users) {  // Using forEach() to iterate over each user  users.forEach(user => {  // Check if the user has an unread notification  if (user.hasUnreadNotification) {  // Send a notification to the user  console.log(`Notification sent to ${user.name}`);  }  });  }  // Sending notifications  sendNotifications(users);  Alternative solution using loop:  // Sample data: Array of users  const users = [  { name: "John", hasUnreadNotification: true },  { name: "Jane", hasUnreadNotification: false },  { name: "Doe", hasUnreadNotification: true }  ];  // Function to send notifications to users with unread notifications using a loop  function sendNotifications(users) {  // Loop through each user in the users array  for (let i = 0; i < users.length; i++) {  // Check if the user has an unread notification  if (users[i].hasUnreadNotification) {  // Send a notification to the user  console.log(`Notification sent to ${users[i].name}`);  }  }  }  // Sending notifications  sendNotifications(users); |

## Exercise 4: Calculate Total Sales

**Problem Statement**:

Given an array of sales, each with a productName and a quantity, calculate the total number of products sold.

|  |
| --- |
| const sales = [  { productName: "Laptop", quantity: 5 },  { productName: "Mouse", quantity: 10 },  { productName: "Keyboard", quantity: 7 } ]; |

**Code Template**:

|  |
| --- |
| let totalSales = 0; function calculateTotalSales(sales) {  // Use forEach() to calculate the total number of products sold } calculateTotalSales(sales); console.log(totalSales); |

**Expected Output**:

22

Solution:

// Sample data: Array of sales

const sales = [

{ productName: "Laptop", quantity: 5 },

{ productName: "Mouse", quantity: 10 },

{ productName: "Keyboard", quantity: 7 }

];

// Initialize total sales to 0

let totalSales = 0;

// Function to calculate the total number of products sold

function calculateTotalSales(sales) {

// Using forEach() to iterate over each sale

sales.forEach(sale => {

// Add the quantity of each sale to the total sales

totalSales += sale.quantity;

});

}

// Calculate the total sales

calculateTotalSales(sales);

console.log(totalSales); // Output: 22

Alternative solution using loop:

// Sample data: Array of sales

const sales = [

{ productName: "Laptop", quantity: 5 },

{ productName: "Mouse", quantity: 10 },

{ productName: "Keyboard", quantity: 7 }

];

// Initialize total sales to 0

let totalSales = 0;

// Function to calculate the total number of products sold using a loop

function calculateTotalSales(sales) {

// Loop through each sale in the sales array

for (let i = 0; i < sales.length; i++) {

// Add the quantity of each sale to the total sales

totalSales += sales[i].quantity;

}

}

// Calculate the total sales

calculateTotalSales(sales);

console.log(totalSales); // Output: 22

# some()

## Exercise 1: Check for VIP Users

**Problem Statement**:

Given an array of users, each with a name and a isVIP boolean property, check if there's at least one VIP user in the list.

const users = [

{ name: "John", isVIP: false },

{ name: "Jane", isVIP: true },

{ name: "Doe", isVIP: false }

];

**Code Template**:

function hasVIPUser(users) {

// Use some() to check if there's at least one VIP user

}

const hasVIP = hasVIPUser(users);

console.log(hasVIP);

**Expected Output**:

True

Solution:

// Define an array of users with name and isVIP properties

const users = [

{ name: 'John', isVIP: false },

{ name: 'Jane', isVIP: true },

{ name: 'Doe', isVIP: false },

]

function hasVIPUser(users) {

// The some() method tests whether at least one element in the array passes

// the test implemented by the provided function. Here, we're using some()

// to check if there's at least one user with the isVIP property set to true.

return users.some((user) => {

return user.isVIP === true

})

}

// Call the hasVIPUser function with the users array as its argument and store the result in hasVIP

const hasVIP = hasVIPUser(users)

// Print the result to the console

console.log(hasVIP) // True

Alternative solution using loop:

// Define an array of users with name and isVIP properties

const users = [

{ name: "John", isVIP: false },

{ name: "Jane", isVIP: true },

{ name: "Doe", isVIP: false }

];

function hasVIPUser(users) {

// Use a traditional for loop to iterate over each user in the array

for (let i = 0; i < users.length; i++) {

// If the isVIP property of the current user is true

if (users[i].isVIP === true) {

// Return true immediately as we found a VIP user

return true;

}

}

// If the loop completes without finding a VIP user, return false

return false;

}

// Call the hasVIPUser function with the users array as its argument and store the result in hasVIP

const hasVIP = hasVIPUser(users);

// Print the result to the console

console.log(hasVIP); // True

## Exercise 2: Check for Out of Stock Products

**Problem Statement**:

Given an array of products, each with a name and a stockQuantity, check if any product is out of stock (stockQuantity is 0).

const products = [

{ name: "Laptop", stockQuantity: 5 },

{ name: "Mouse", stockQuantity: 0 },

{ name: "Keyboard", stockQuantity: 7 }

];

**Code Template**:

function hasOutOfStockProduct(products) {

// Use some() to check if any product is out of stock

}

const isOutOfStock = hasOutOfStockProduct(products);

console.log(isOutOfStock);

**Expected Output**:

true

Solution:

// Sample data: Array of products

const products = [

{ name: "Laptop", stockQuantity: 5 },

{ name: "Mouse", stockQuantity: 0 },

{ name: "Keyboard", stockQuantity: 7 }

];

// Function to check if any product is out of stock

function hasOutOfStockProduct(products) {

// Using some() to check if any product has a stock quantity of 0

return products.some(product => product.stockQuantity === 0);

}

// Check if any product is out of stock

const isOutOfStock = hasOutOfStockProduct(products);

console.log(isOutOfStock); // Output: true

Alternative solution using loop:

// Sample data: Array of products

const products = [

{ name: "Laptop", stockQuantity: 5 },

{ name: "Mouse", stockQuantity: 0 },

{ name: "Keyboard", stockQuantity: 7 }

];

// Function to check if any product is out of stock using a loop

function hasOutOfStockProduct(products) {

// Loop through each product in the products array

for (let i = 0; i < products.length; i++) {

// Check if the stock quantity of the product is 0

if (products[i].stockQuantity === 0) {

// Return true if an out-of-stock product is found

return true;

}

}

// Return false if no out-of-stock products are found

return false;

}

// Check if any product is out of stock

const isOutOfStock = hasOutOfStockProduct(products);

console.log(isOutOfStock); // Output: true

## Exercise 3: Check for Users from a Specific Country

**Problem Statement**:

Given an array of users, each with a name and a country, check if there's at least one user from "Canada".

const users = [

{ name: "John", country: "USA" },

{ name: "Jane", country: "Canada" },

{ name: "Doe", country: "UK" }

];

**Code Template**:

function hasUserFromCanada(users) {

// Use some() to check if there's at least one user from "Canada"

}

const hasCanadian = hasUserFromCanada(users);

console.log(hasCanadian);

**Expected Output**:

True

Solution:

// Sample data: Array of users

const users = [

{ name: "John", country: "USA" },

{ name: "Jane", country: "Canada" },

{ name: "Doe", country: "UK" }

];

// Function to check if there's at least one user from "Canada"

function hasUserFromCanada(users) {

// Using some() to check for at least one user from "Canada"

return users.some(user => user.country === "Canada");

}

// Check if there's a user from Canada

const hasCanadian = hasUserFromCanada(users);

console.log(hasCanadian); // Output: true

Alternative solution using loop:

// Sample data: Array of users

const users = [

{ name: 'John', country: 'USA' },

{ name: 'Jane', country: 'Canada' },

{ name: 'Doe', country: 'UK' },

]

// Function to check if there's at least one user from "Canada" using a loop

function hasUserFromCanada(users) {

// Loop through each user in the users array

for (let i = 0; i < users.length; i++) {

// Check if the user's country is "Canada"

if (users[i].country === 'Canada') {

// Return true if a user from Canada is found

return true

}

}

// Return false if no users from Canada are found

return false

}

// Check if there's a user from Canada using a loop

const hasCanadian = hasUserFromCanada(users)

console.log(hasCanadian) // Output: true

Exercise 4: Check for Expired Products

**Problem Statement**:

Given an array of products, each with a name and an expiryDate (in the format "YYYY-MM-DD"), check if any product has expired (current date is greater than expiry date).

const products = [

{ name: "Milk", expiryDate: "2023-01-01" },

{ name: "Bread", expiryDate: "2022-12-01" },

{ name: "Butter", expiryDate: "2023-02-15" }

];

**Code Template**:

function hasExpiredProduct(products) {

// Use some() to check if any product has expired

}

const hasExpired = hasExpiredProduct(products);

console.log(hasExpired);

**Expected Output** (depends on the current date):

true or false

Solution:

// Sample data: Array of products

const products = [

{ name: "Milk", expiryDate: "2023-01-01" },

{ name: "Bread", expiryDate: "2022-12-01" },

{ name: "Butter", expiryDate: "2023-02-15" }

];

// Function to check if any product has expired

function hasExpiredProduct(products) {

// Get the current date

const currentDate = new Date();

// Using some() to check for any expired product

return products.some(product => {

// Convert the expiry date string to a Date object

const expiryDate = new Date(product.expiryDate);

// Check if the expiry date is before the current date

return expiryDate < currentDate;

});

}

// Check if any product has expired

const hasExpired = hasExpiredProduct(products);

console.log(hasExpired); // Output: true or false (depends on the current date)

Alternative solution using loop:

// Sample data: Array of products

const products = [

{ name: 'Milk', expiryDate: '2023-01-01' },

{ name: 'Bread', expiryDate: '2022-12-01' },

{ name: 'Butter', expiryDate: '2023-02-15' },

]

// Function to check if any product has expired using a loop

function hasExpiredProduct(products) {

const currentDate = new Date()

for (let i = 0; i < products.length; i++) {

const expiryDate = new Date(products[i].expiryDate)

if (expiryDate < currentDate) {

return true // An expired product is found

}

}

return false // No expired products found

}

const hasExpired = hasExpiredProduct(products)

console.log(hasExpired) // Output: true or false (depends on the current date)

# every()

## Exercise 1: Check if All Users are Active

**Problem Statement**:

Given an array of users, each with a name and an isActive boolean property, check if all users are active.

const users = [

{ name: "John", isActive: true },

{ name: "Jane", isActive: true },

{ name: "Doe", isActive: true }

];

**Code Template**:

function areAllUsersActive(users) {

// Use every() to check if all users are active

}

const allActive = areAllUsersActive(users);

console.log(allActive);

**Expected Output**:

true

Solution:

// Sample data with users and their isActive status

const users = [

{ name: 'John', isActive: true },

{ name: 'Jane', isActive: true },

{ name: 'Doe', isActive: true },

]

// Function to check if all users are active

function areAllUsersActive(users) {

// The every() method checks if all elements in the array satisfy the condition provided by the callback function

return users.every(function (user) {

// Return the value of the isActive property for each user

// If any user is not active (isActive is false), the every() method will return false

// If all users are active (isActive is true for all), the every() method will return true

return user.isActive

})

}

// Call the areAllUsersActive function with the users array as its argument and store the result in allActive

const allActive = areAllUsersActive(users)

// Print the result to the console

console.log(allActive) // Expected: true

Alternative solution with loop:

// Sample data with users and their isActive status

const users = [

{ name: "John", isActive: true },

{ name: "Jane", isActive: true },

{ name: "Doe", isActive: true }

];

// Function to check if all users are active using a loop

function areAllUsersActive(users) {

// Use a traditional for loop to iterate over each user in the array

for (let i = 0; i < users.length; i++) {

// If the isActive property of the current user is false

if (users[i].isActive === false) {

// Return false immediately as we found an inactive user

return false;

}

}

// If the loop completes without finding an inactive user, return true

return true;

}

// Call the areAllUsersActive function with the users array as its argument and store the result in allActive

const allActive = areAllUsersActive(users);

// Print the result to the console

console.log(allActive); // Expected: true

## Exercise 2: Check if All Products are In Stock

**Problem Statement**:

Given an array of products, each with a name and a stockQuantity, check if all products have a stock quantity greater than 0.

const products = [

{ name: "Laptop", stockQuantity: 5 },

{ name: "Mouse", stockQuantity: 3 },

{ name: "Keyboard", stockQuantity: 7 }

];

**Code Template**:

function areAllProductsInStock(products) {

// Use every() to check if all products are in stock

}

const allInStock = areAllProductsInStock(products);

console.log(allInStock);

**Expected Output**:

True

Solution:

// Sample data: Array of products

const products = [

{ name: "Laptop", stockQuantity: 5 },

{ name: "Mouse", stockQuantity: 3 },

{ name: "Keyboard", stockQuantity: 7 }

];

// Function to check if all products are in stock

function areAllProductsInStock(products) {

// Using every() to check if every product's stockQuantity is greater than 0

return products.every(product => product.stockQuantity > 0);

}

// Check if all products are in stock

const allInStock = areAllProductsInStock(products);

console.log(allInStock); // Output: true

Alternative solution using loop:  
// Sample data: Array of products (same as above)

const products = [

{ name: "Laptop", stockQuantity: 5 },

{ name: "Mouse", stockQuantity: 3 },

{ name: "Keyboard", stockQuantity: 7 }

];

// Function to check if all products are in stock using a loop

function areAllProductsInStock(products) {

// Loop through each product in the products array

for (let i = 0; i < products.length; i++) {

// Check if the stock quantity of the product is not greater than 0

if (products[i].stockQuantity <= 0) {

return false; // A product is out of stock

}

}

return true; // All products are in stock

}

// Check if all products are in stock using a loop

const allInStock = areAllProductsInStock(products);

console.log(allInStock); // Output: true

## Exercise 3: Check if All Users are Above 18

**Problem Statement**:

Given an array of users, each with a name and an age, check if all users are above 18 years old.

const users = [

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

{ name: "Jane", age: 20 },

{ name: "Doe", age: 19 }

];

**Code Template**:

function areAllUsersAdult(users) {

// Use every() to check if all users are above 18

}

const allAdults = areAllUsersAdult(users);

console.log(allAdults);

**Expected Output**:

true

Solution:

// Sample data: Array of users

const users = [

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

{ name: "Jane", age: 20 },

{ name: "Doe", age: 19 }

];

// Function to check if all users are adults (above 18)

function areAllUsersAdult(users) {

// Using every() to check if every user's age is greater than 18

return users.every(user => user.age > 18);

}

// Check if all users are adults

const allAdults = areAllUsersAdult(users);

console.log(allAdults); // Output: true

Alternative solution using loop:

// Sample data: Array of users (same as above)

const users = [

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

{ name: "Jane", age: 20 },

{ name: "Doe", age: 19 }

];

// Function to check if all users are adults using a loop

function areAllUsersAdult(users) {

// Loop through each user in the users array

for (let i = 0; i < users.length; i++) {

// Check if the user's age is not greater than 18

if (users[i].age <= 18) {

return false; // A user is not an adult

}

}

return true; // All users are adults

}

// Check if all users are adults using a loop

const allAdults = areAllUsersAdult(users);

console.log(allAdults); // Output: true

String Methods

# charAt()

## Exercise 1: Extract Initials from Name

**Problem Statement**:

Given a name in the format "FirstName LastName", extract the initials using the charAt() method.

const name = "John Doe";

**Code Template**:

function getInitials(name) {

// Use charAt() to extract initials from the name

}

const initials = getInitials(name);

console.log(initials);

**Expected Output**:

JD

Solution:

// Sample data: A name

const name = 'John Doe'

// Function to extract initials using charAt()

function getInitials(name) {

// Split the name into its components

const parts = name.split(' ')

// Extract the first character of each part

return parts[0].charAt(0) + parts[1].charAt(0)

}

// Get the initials

const initials = getInitials(name)

console.log(initials) // Output: JD

Alternative solution using loop:

// Sample data: A name

const name = "John Doe";

// Function to extract initials using a loop

function getInitials(name) {

// Split the name into parts

const parts = name.split(" ");

// Initialize an empty string for the initials

let initials = "";

// Loop through each part of the name

for (let i = 0; i < parts.length; i++) {

// Add the first character of each part to the initials

initials += parts[i].charAt(0);

}

// Return the initials

return initials;

}

// Get the initials

const initials = getInitials(name);

console.log(initials); // Output: JD

## Exercise 2: Check First Letter of a Username

**Problem Statement**:

Given a username, check if the first letter is an uppercase letter using the charAt() method.

const username = "Admin123";

**Code Template**:

function isFirstLetterUppercase(username) {

// Use charAt() to check the first letter of the username

}

const isUppercase = isFirstLetterUppercase(username);

console.log(isUppercase);

**Expected Output**:

true

Solution:

// Sample data: A username

const username = "Admin123";

// Function to check if the first letter of the username is uppercase

function isFirstLetterUppercase(username) {

// Get the first character of the username

const firstChar = username.charAt(0);

// Check if the first character is the same as its uppercase version

return firstChar === firstChar.toUpperCase();

}

// Check if the first letter is uppercase

const isUppercase = isFirstLetterUppercase(username);

console.log(isUppercase); // Output: true

Alternative solution using loop:

// Sample data: A username

const username = "Admin123";

// Function to check if the first letter of the username is uppercase using a loop

function isFirstLetterUppercase(username) {

// Loop through the first character of the username

for (let i = 0; i < 1; i++) {

// Check if the first character is the same as its uppercase version

return username[i] === username[i].toUpperCase();

}

}

// Check if the first letter is uppercase

const isUppercase = isFirstLetterUppercase(username);

console.log(isUppercase); // Output: true

## Exercise 3: Extract File Extension

**Problem Statement**:

Given a filename, extract the last character to determine the file type using the charAt() method.

const filename = "document.txt";

**Code Template**:

function getFileType(filename) {

// Use charAt() to extract the last character of the filename

}

const fileType = getFileType(filename);

console.log(fileType);

**Expected Output**:

t

Solution:

// Sample data: A filename

const filename = "document.txt";

// Function to extract the last character of the filename

function getFileType(filename) {

// Use charAt() with length - 1 to get the last character

return filename.charAt(filename.length - 1);

}

// Get the file type

const fileType = getFileType(filename);

console.log(fileType); // Output: t

Alternative solution using loop:

// Sample data: A filename

const filename = 'document.txt'

function getFileType(filename) {

// Initialize a variable to store the last character

let lastChar = ''

// Loop through each character in the filename

for (let i = 0; i < filename.length; i++) {

// Assign the current character to lastChar

lastChar = filename[i]

}

// Return the last character

return lastChar

}

// Get the file type

const fileType = getFileType(filename)

console.log(fileType) // Output: t

# indexOf()

## Exercise 1: Check if Email Contains '@' Symbol

**Problem Statement**:

Given an email address, check if it contains the '@' symbol using the indexOf() method.

const email = "john.doe@example.com";

**Code Template**:

function isValidEmail(email) {

// Use indexOf() to check if email contains '@' symbol

}

const isValid = isValidEmail(email);

console.log(isValid);

**Expected Output**:

true

Solution:

// Sample data: An email address

const email = "john.doe@example.com";

// Function to check if the email contains the '@' symbol

function isValidEmail(email) {

// Use indexOf() to find the index of '@' in the email

return email.indexOf('@') !== -1;

}

// Check if the email is valid

const isValid = isValidEmail(email);

console.log(isValid); // Output: true

Alternative solution using loop:

// Sample data: An email address

const email = 'john.doe@example.com'

// Function to check if the email contains the '@' symbol using a loop

function isValidEmail(email) {

// Loop through each character in the email

for (let i = 0; i < email.length; i++) {

// Check if the current character is '@'

if (email[i] === '@') {

return true // '@' symbol is found

}

}

return false // '@' symbol is not found

}

// Check if the email is valid

const isValid = isValidEmail(email)

console.log(isValid) // Output: true

## Exercise 2: Find Position of Substring

**Problem Statement**:

Given a sentence, find the position of the word "JavaScript" using the indexOf() method.

const sentence = "I love programming in JavaScript!";

**Code Template**:

function findPosition(sentence) {

// Use indexOf() to find the position of "JavaScript"

}

const position = findPosition(sentence);

console.log(position);

**Expected Output**:

22

Solution:

// Sample data: A sentence

const sentence = "I love programming in JavaScript!";

// Function to find the position of "JavaScript" in the sentence

function findPosition(sentence) {

// Use indexOf() to find the index of "JavaScript"

return sentence.indexOf("JavaScript");

}

// Find the position of "JavaScript"

const position = findPosition(sentence);

console.log(position); // Output: 22

Alternative solution using loop:

// Sample data: A sentence

const sentence = 'I love programming in JavaScript!'

// Function to find the position of "JavaScript" in the sentence using a loop

function findPosition(sentence) {

// The word to find

const word = 'JavaScript'

// Loop through each character in the sentence

for (let i = 0; i <= sentence.length - word.length; i++) {

// Check if the substring from the current position matches "JavaScript"

if (sentence.substring(i, i + word.length) === word) {

return i // Position of "JavaScript" found

}

}

return -1 // "JavaScript" not found

}

// Find the position of "JavaScript"

const position = findPosition(sentence)

console.log(position) // Output: 22

## Exercise 3: Check Protocol of a URL

**Problem Statement**:

Given a URL, check if it uses the "https" protocol using the indexOf() method.

const url = "https://www.example.com";

**Code Template**:

function usesHttps(url) {

// Use indexOf() to check if URL starts with "https"

}

const isHttps = usesHttps(url);

console.log(isHttps);

**Expected Output**:

true

Solution:

// Sample data: A URL

const url = "https://www.example.com";

// Function to check if the URL uses "https" protocol

function usesHttps(url) {

// Check if the URL starts with "https"

return url.indexOf("https://") === 0;

}

// Check if the URL is https

const isHttps = usesHttps(url);

console.log(isHttps); // Output: true

Alternative solution using loop:

// Sample data: A URL (same as above)

const url = "https://www.example.com";

// Function to check if the URL uses "https" protocol using a loop

function usesHttps(url) {

const protocol = "https://";

for (let i = 0; i < protocol.length; i++) {

if (url[i] !== protocol[i]) {

return false; // The URL does not start with "https"

}

}

return true; // The URL starts with "https"

}

// Check if the URL is https using a loop

const isHttps = usesHttps(url);

console.log(isHttps); // Output: true

# lastIndexOf()

## Exercise 1: Find Last Occurrence of a Word

**Problem Statement**:

Given a sentence, find the last occurrence of the word "apple" using the lastIndexOf() method.

const sentence = "I ate an apple. Then I bought another apple.";

**Code Template**:

function findLastOccurrence(sentence) {

// Use lastIndexOf() to find the last occurrence of "apple"

}

const lastPosition = findLastOccurrence(sentence);

console.log(lastPosition);

**Expected Output**:

38

Solution:

// Sample data: A sentence

const sentence = "I ate an apple. Then I bought another apple.";

// Function to find the last occurrence of "apple"

function findLastOccurrence(sentence) {

// Using lastIndexOf() to find the last index of "apple"

return sentence.lastIndexOf("apple");

}

// Finding the last position of "apple"

const lastPosition = findLastOccurrence(sentence);

console.log(lastPosition); // Expected Output: 38

Alternative solution using loop:

// Sample data: A sentence

const sentence = "I ate an apple. Then I bought another apple.";

// Function to find the last occurrence of "apple" using a loop

function findLastOccurrence(sentence) {

let lastPosition = -1;

// Loop through the sentence

for (let i = 0; i <= sentence.length - "apple".length; i++) {

// Extract the substring from the current position

const substring = sentence.substring(i, i + "apple".length);

// Check if the substring matches "apple"

if (substring === "apple") {

// Update the last position of "apple"

lastPosition = i;

}

}

return lastPosition;

}

// Finding the last position of "apple"

const lastPosition = findLastOccurrence(sentence);

console.log(lastPosition); // Expected Output: 38

## Exercise 2: Check File Extension

**Problem Statement**:

Given a filename, extract its extension using the lastIndexOf() method.

const filename = "document.pdf";

**Code Template**:

function getFileExtension(filename) {

// Use lastIndexOf() to extract the file extension

}

const extension = getFileExtension(filename);

console.log(extension);

**Expected Output**:

pdf

Solution:

// Sample data: A filename

const filename = "document.pdf";

// Function to extract the file extension

function getFileExtension(filename) {

// Find the last index of '.'

const index = filename.lastIndexOf('.');

// Extract the substring from after the '.' to the end

// If there is no '.', return an empty string

return index !== -1 ? filename.substring(index + 1) : '';

}

// Get the file extension

const extension = getFileExtension(filename);

console.log(extension); // Expected Output: pdf

Alternative solution using loop:

// Sample data: A filename

const filename = 'document.pdf'

function getFileExtension(filename) {

let extension = ''

let i = filename.length - 1

// Loop backwards through the filename

while (i >= 0 && filename[i] !== '.') {

// Prepend the current character to the extension

extension = filename[i] + extension

i--

}

return extension

}

// Get the file extension

const extension = getFileExtension(filename)

console.log(extension) // Expected Output: pdf

## Exercise 3: Extract Domain from Email

**Problem Statement**:

Given an email address, extract the domain using the lastIndexOf() method.

const email = "john.doe@example.com";

**Code Template**:

function getEmailDomain(email) {

// Use lastIndexOf() to extract the domain from the email

}

const domain = getEmailDomain(email);

console.log(domain);

**Expected Output**:

example.com

Solution:

// Sample data: An email address

const email = "john.doe@example.com";

// Function to extract the domain from the email

function getEmailDomain(email) {

// Find the last index of '@'

const index = email.lastIndexOf('@');

// Extract the substring from after the '@' to the end

return index !== -1 ? email.substring(index + 1) : '';

}

// Get the email domain

const domain = getEmailDomain(email);

console.log(domain); // Expected Output: example.com

Alternative solution using loop:

// Sample data: An email address

const email = 'john.doe@example.com'

// Function to extract the domain from the email

function getEmailDomain(email) {

let domain = ''

let i = email.length - 1

// Loop backwards through the email

while (i >= 0) {

if (email[i] === '@') {

// Once '@' is found, break the loop

break

}

// Prepend the current character to the domain

domain = email[i] + domain

i--

}

return domain

}

// Get the email domain

const domain = getEmailDomain(email)

console.log(domain) // Expected Output: example.com

# slice()

## Exercise 1: Extract Username from Email

**Problem Statement**:

Given an email address, extract the username using the slice() method.

|  |
| --- |
| const email = "john.doe@example.com"; |

**Code Template**:

|  |
| --- |
| function getUsername(email) {  // Use slice() to extract the username from the email } const username = getUsername(email); console.log(username); |

**Expected Output**:

|  |
| --- |
| John.doe |

Solution:

|  |
| --- |
| **function** **getUserName**(email) {  **const** indexOfAt = email.indexOf('@')  **const** userName = email.slice(0, indexOfAt)  **return** userName } **const** email = 'john.doe@example.com' **const** userName = getUserName(email) console.log(userName) |

**Alternative solutions:**

*Solution using ‘For’ loop:*

|  |
| --- |
| **function** **getUserName**(email) {  **let** userName = ''  **for** (**let** i = 0; i < email.length; i++) {  **if** (email[i] === '@') {  **break**  }  userName += email[i]  }  **return** userName } **const** email = 'john.doe@example.com' **const** userName = getUserName(email) console.log(userName) |

*Solution using ‘Array’ and ‘For’ loop:*

|  |
| --- |
| **function** **getUserName**(email) {  **let** userName = ''  **const** emailArray = email.split('')  **for** (**let** i = 0; i < emailArray.length; i++) {  **if** (emailArray[i] === '@') {  **break**  }  userName += emailArray[i]  }  **return** userName } **const** email = 'john.doe@example.com' **const** userName = getUserName(email) console.log(userName) |

## Exercise 2: Get First Name from Full Name

**Problem Statement**:

Given a full name, extract the first name using the slice() method.

|  |
| --- |
| **const** fullName = "John Doe"; |

**Code Template**:

|  |
| --- |
| function getFirstName(fullName) {  *// Use slice() to extract the first name from the full name* } const firstName = getFirstName(fullName); console.log(firstName); |

**Expected Output**:

John

**Solution:**

|  |
| --- |
| *// Define a string constant named 'fullName' with the value 'John Doe'* **const** fullName = 'John Doe'  *// Define a function named 'getFirstName' that takes a parameter 'fullName'* **function** **getFirstName**(fullName) {  *// Use the 'indexOf' method to find the position of the first space character in 'fullName'*  *// and store it in a constant named 'indexOfSpace'*  **const** indexOfSpace = fullName.indexOf(' ')   *// Use the 'slice' method to extract a substring from 'fullName'*  *// that starts at index 0 and ends at 'indexOfSpace',*  *// and store it in a constant named 'firstName'*  **const** firstName = fullName.slice(0, indexOfSpace)   *// Return the 'firstName' string as the result of the function*  **return** firstName } |

Alternative Solutions:

Solution using ‘For’ loop:

|  |
| --- |
| Define a string constant named 'fullName' **with** the value 'John Doe' **const** fullName = 'John Doe';  *// Define a function named 'getFirstName' that takes a parameter 'fullName'* **function** **getFirstName**(fullName) {  *// Initialize an empty string variable 'firstName' to store the first name*  **let** firstName = '';   *// Use a 'for' loop to iterate through each character of the 'fullName' string*  **for** (**let** i = 0; i < fullName.length; i++) {  *// Check if the current character is a space (' ')*  **if** (fullName[i] === ' ') {  *// If it is, break out of the loop early,*  *// because we've reached the end of the first name*  **break**;  }  *// If it's not a space, concatenate (add) the current character to 'firstName'*  firstName += fullName[i];  }  *// Return the 'firstName' string as the result of the function*  **return** firstName; }  *// Call the 'getFirstName' function with 'fullName' as an argument and* *// store the result in a new constant named 'firstName'* **const** firstName = getFirstName(fullName);  *// Output the 'firstName' to the console* console.log(firstName); |

## Exercise 3: Extract Protocol from URL

**Problem Statement**:

Given a URL, extract the protocol (e.g., "http", "https") using the slice() method.

|  |
| --- |
| **const** url = "https://www.example.com"; |

**Code Template**:

|  |
| --- |
| **function** **getProtocol**(url) {  *// Use slice() to extract the protocol from the URL* } **const** protocol = getProtocol(url); console.log(protocol); |

**Expected Output**:

https

# split()

#### Exercise 1: Split Sentence into Words

Problem Statement:  
Given a sentence, split it into an array of words using the split() method.

const sentence = "JavaScript is fun";

function splitSentence(sentence) {

// Use split() to create an array of words

}

const words = splitSentence(sentence);

console.log(words);

Expected Output:  
["JavaScript", "is", "fun"]

Solution:

// Sample data: A sentence

const sentence = "JavaScript is fun";

// Function to split the sentence into an array of words

function splitSentence(sentence) {

// Splitting the sentence on spaces

return sentence.split(' ');

}

// Splitting the sentence into words

const words = splitSentence(sentence);

console.log(words); // Expected Output: ["JavaScript", "is", "fun"]

Alternative solution using loop:

// Sample data: A sentence

const sentence = "JavaScript is fun";

// Function to split the sentence into an array of words using a loop

function splitSentence(sentence) {

let words = [];

let currentWord = "";

// Loop through each character in the sentence

for (let i = 0; i < sentence.length; i++) {

if (sentence[i] === ' ' || i === sentence.length - 1) {

// Add the last character to the current word if it's the end of the sentence

if (i === sentence.length - 1) {

currentWord += sentence[i];

}

// Add the current word to the words array and reset currentWord

if (currentWord.length > 0) {

words.push(currentWord);

currentWord = "";

}

} else {

// Build the current word

currentWord += sentence[i];

}

}

return words;

}

// Splitting the sentence into words

const words = splitSentence(sentence);

console.log(words); // Expected Output: ["JavaScript", "is", "fun"]

#### **[Exercise 2: Split CSV Line](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-split-csv-line)**

Problem Statement:  
Given a string representing a line from a CSV file, split it into an array using the split() method.

const csvLine = "John,Doe,30,Male,Engineer";

function splitCsvLine(line) {

// Use split() to create an array from the CSV line

}

const values = splitCsvLine(csvLine);

console.log(values);

Expected Output:  
["John", "Doe", "30", "Male", "Engineer"]

Solution:

// Sample data: A line from a CSV file

const csvLine = "John,Doe,30,Male,Engineer";

// Function to split the CSV line into an array

function splitCsvLine(line) {

// Splitting the line on commas

return line.split(',');

}

// Splitting the CSV line into values

const values = splitCsvLine(csvLine);

console.log(values); // Expected Output: ["John", "Doe", "30", "Male", "Engineer"]

Alternative solution using loop:

// Sample data: A line from a CSV file

const csvLine = "John,Doe,30,Male,Engineer";

// Function to manually parse the CSV line into an array

function splitCsvLine(line) {

let values = [];

let currentElement = "";

// Loop through each character in the line

for (let i = 0; i < line.length; i++) {

if (line[i] === ',' || i === line.length - 1) {

// Append the last character if it's the end of the line

if (i === line.length - 1) {

currentElement += line[i];

}

// Add the current element to the values array

values.push(currentElement);

// Reset currentElement for the next element

currentElement = "";

} else {

// Build the current element

currentElement += line[i];

}

}

return values;

}

// Parsing the CSV line into values

const values = splitCsvLine(csvLine);

console.log(values); // Expected Output: ["John", "Doe", "30", "Male", "Engineer"]

#### **[Exercise 3: Split URL Parameters](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-split-url-parameters)**

Problem Statement:  
Given a URL query string, split it into key-value pairs using the split() method.

const queryString = "name=John&age=30&gender=Male";

function splitQueryString(query) {

// Use split() to create an array of key-value pairs

}

const parameters = splitQueryString(queryString);

console.log(parameters);

Expected Output:  
["name=John", "age=30", "gender=Male"]

Solution:

// Sample data: A URL query string

const queryString = "name=John&age=30&gender=Male";

// Function to split the query string into an array of key-value pairs

function splitQueryString(query) {

// Splitting the query string on '&'

return query.split('&');

}

// Splitting the query string into parameters

const parameters = splitQueryString(queryString);

console.log(parameters); // Expected Output: ["name=John", "age=30", "gender=Male"]

Alternative solution using loop:

// Sample data: A URL query string

const queryString = 'name=John&age=30&gender=Male'

// Function to manually parse the query string into an array of key-value pairs

function splitQueryString(query) {

let parameters = []

let currentPair = ''

// Loop through each character in the query string

for (let i = 0; i < query.length; i++) {

if (query[i] === '&' || i === query.length - 1) {

// Append the last character if it's the end of the query string

if (i === query.length - 1) {

currentPair += query[i]

}

// Add the current pair to the parameters array

parameters.push(currentPair)

// Reset currentPair for the next key-value pair

currentPair = ''

} else {

// Build the current key-value pair

currentPair += query[i]

}

}

return parameters

}

// Parsing the query string into parameters

const parameters = splitQueryString(queryString)

console.log(parameters) // Expected Output: ["name=John", "age=30", "gender=Male"]

# [replace()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "replace)

#### **[Exercise 1: Replace Placeholder](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-replace-placeholder)**

Problem Statement:  
Given a template string, replace the placeholder {name} with a given name using the replace() method.

const template = "Hello, {name}!";

const name = "John";

function replacePlaceholder(template, name) {

// Use replace() to substitute {name} with the actual name

}

const greeting = replacePlaceholder(template, name);

console.log(greeting);

Expected Output:  
Hello, John!

Solution:

// Sample data: A template string and a name

const template = "Hello, {name}!";

const name = "John";

// Function to replace the placeholder with the given name

function replacePlaceholder(template, name) {

// Using replace() to substitute {name} with the actual name

return template.replace("{name}", name);

}

// Replacing the placeholder and creating the greeting

const greeting = replacePlaceholder(template, name);

console.log(greeting); // Expected Output: Hello, John!

Alternative solution using loop:

// Sample data: A template string and a name

const template = 'Hello, {name}!'

const name = 'John'

// Function to manually replace the placeholder using a loop

function replacePlaceholder(template, name) {

let result = '' // Initialize an empty string to build the result

let isPlaceholder = false // Flag to track if we are within the placeholder

// Loop through each character in the template string

for (let i = 0; i < template.length; i++) {

if (template[i] === '{') {

// When we find an opening brace, set the flag to true

isPlaceholder = true

// Append the name to the result, replacing the placeholder

result += name

} else if (template[i] === '}') {

// When we find a closing brace, set the flag to false

isPlaceholder = false

continue // Skip adding the closing brace to the result

} else if (!isPlaceholder) {

// If not currently in a placeholder, append the character to the result

result += template[i]

}

// If we are inside a placeholder, we skip adding characters until we find the closing brace

}

return result // Return the final string with the placeholder replaced

}

// Creating the greeting by manually replacing the placeholder

const greeting = replacePlaceholder(template, name)

console.log(greeting) // Expected Output: Hello, John!

#### **[Exercise 2: Replace Spaces with Dashes](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-replace-spaces-with-dashes)**

Problem Statement:  
Given a string, replace all spaces with dashes using the replace() method.

const string = "JavaScript is fun";

function replaceSpaces(string) {

// Use replace() to substitute spaces with dashes

}

const dashedString = replaceSpaces(string);

console.log(dashedString);

Expected Output:  
JavaScript-is-fun

Solution:

// Sample data: A string

const string = "JavaScript is fun";

// Function to replace all spaces with dashes

function replaceSpaces(string) {

// Using replace() with a regular expression to match all spaces

// The '/ /g' is a regular expression where:

// ' ' (space) is the pattern to match

// 'g' is the global modifier to match all occurrences of the pattern

return string.replace(/ /g, '-');

}

// Replace spaces with dashes

const dashedString = replaceSpaces(string);

console.log(dashedString); // Expected Output: JavaScript-is-fun

Alternative solution using loop:

// Sample data: A string

const string = "JavaScript is fun";

// Function to replace all spaces with dashes using a loop

function replaceSpaces(string) {

let result = ""; // Initialize an empty string to build the result

// Loop through each character in the string

for (let i = 0; i < string.length; i++) {

if (string[i] === ' ') {

// If the current character is a space, append a dash to the result

result += '-';

} else {

// Otherwise, append the current character to the result

result += string[i];

}

}

return result; // Return the final string with spaces replaced by dashes

}

// Replace spaces with dashes

const dashedString = replaceSpaces(string);

console.log(dashedString); // Expected Output: JavaScript-is-fun

#### **[Exercise 3: Replace Domain in Email Address](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-replace-domain-in-email-address)**

Problem Statement:  
Given an email address, replace the domain with a new domain using the replace() method.

const email = "john.doe@example.com";

const newDomain = "newdomain.com";

function replaceDomain(email, newDomain) {

// Use replace() to substitute the old domain with the new domain

}

const newEmail = replaceDomain(email, newDomain);

console.log(newEmail);

Expected Output:  
john.doe@newdomain.com

Solution:

// Sample data: An email address and a new domain

const email = "john.doe@example.com";

const newDomain = "newdomain.com";

// Function to replace the domain of an email address

function replaceDomain(email, newDomain) {

// Extract the local part of the email (before the '@')

const localPart = email.split('@')[0];

// Use replace() to substitute the entire domain part with the new domain

// Here, replace() is used to replace the original email's domain (after '@') with the newDomain

return localPart + "@" + newDomain;

}

// Replace the domain in the email

const newEmail = replaceDomain(email, newDomain);

console.log(newEmail); // Expected Output: john.doe@newdomain.com

Alternative solution using loop:

// Sample data: An email address and a new domain

const email = "john.doe@example.com";

const newDomain = "newdomain.com";

// Function to manually replace the domain of an email address

function replaceDomain(email, newDomain) {

let atIndex = -1; // Variable to store the index of '@' in the email

// Loop to find the index of '@' in the email

for (let i = 0; i < email.length; i++) {

if (email[i] === '@') {

atIndex = i;

break; // Break the loop once '@' is found

}

}

// If '@' is not found, return the original email

if (atIndex === -1) return email;

// Extract the local part (before '@') and concatenate with the new domain

const localPart = email.substring(0, atIndex);

return localPart + "@" + newDomain;

}

// Replace the domain in the email

const newEmail = replaceDomain(email, newDomain);

console.log(newEmail); // Expected Output: john.doe@newdomain.com

# trim()

#### **Exercise 1: Trim Whitespace**

Problem Statement:  
Given a string with leading and trailing whitespace, remove the whitespace using the trim() method.

const string = " JavaScript is fun ";

function trimWhitespace(string) {

// Use trim() to remove leading and trailing whitespace

}

const trimmedString = trimWhitespace(string);

console.log(trimmedString);

Expected Output:  
JavaScript is fun

Solution:

|  |
| --- |
| *// Define a sample string with leading and trailing spaces.* **const** string = ' JavaScript is fun ';  *// Define a function to trim leading and trailing whitespace from a given string.* **function** **trimWhitespace**(string) {  *// The trim() method returns a new string with both leading and trailing whitespace removed.*  *// It doesn't modify the original string.*  **return** string.trim(); }  *// Call the trimWhitespace function with the sample string and store the result.* **const** trimmedString = trimWhitespace(string);  *// Display the trimmed string in the console.* console.log(trimmedString); |

Alternative Solution with loop:

|  |
| --- |
| *// Define a sample string with leading and trailing spaces.* **const** string = ' Javascript is fun '  *// Define a function to trim leading and trailing whitespace from a given string.* **function** **trimWhitespace**(string) {  *// Initialize startIndex at the beginning of the string.*  **let** startIndex = 0   *// Initialize endIndex at the end of the string.*  **let** endIndex = string.length - 1   *// Increment startIndex as long as the character at the current startIndex is a space.*  *// This loop will stop when it finds the first non-space character or reaches the end of the string.*  **while** (string[startIndex] === ' ') {  startIndex++  }   *// Decrement endIndex as long as the character at the current endIndex is a space.*  *// This loop will stop when it finds the last non-space character or reaches the beginning of the string.*  **while** (string[endIndex] === ' ') {  endIndex--  }   *// Extract the substring from startIndex to endIndex using the slice method.*  *// Since slice does not include the character at the endIndex, we add 1 to include it.*  **const** finalString = string.slice(startIndex, endIndex + 1)   *// Return the trimmed string.*  **return** finalString }  *// Call the trimWhitespace function with the sample string and store the result.* **const** trimmedString = trimWhitespace(string)  *// Display the trimmed string in the console.* console.log(trimmedString) |

#### **[Exercise 2: Log Trimmed Input](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-log-trimmed-input)**

Problem Statement:  
Given a user input string, log the trimmed input using the trim() method.

const userInput = " User input with spaces ";

function logTrimmedInput(input) {

// Use trim() and log the trimmed input

}

const trimmedInput = logTrimmedInput(userInput);

console.log(trimmedInput)

Expected Output:  
User input with spaces

Solution:

|  |
| --- |
| *// Define a string representing user input, which has leading and trailing spaces.* **const** userInput = ' User input with spaces ';  *// Define a function to trim leading and trailing whitespace from a given input.* **function** **logTrimmedInput**(input) {  *// The trim() method returns a new string with both leading and trailing whitespace removed.*  *// It doesn't modify the original string. However, the function is currently always returning the*   *// trimmed version of the `userInput` variable regardless of its `input` parameter.*  **return** userInput.trim(); }  *// Call the logTrimmedInput function with the userInput string and store the result.* **const** trimmedInput = logTrimmedInput(userInput);  *// Display the trimmed userInput string in the console.* console.log(trimmedInput); |

Alternative solution with loop:

|  |
| --- |
| *// Define a string representing user input, which has leading and trailing spaces.* **const** userInput = ' User input with spaces ';  *// Define a function to trim leading and trailing whitespace from a given input.* **function** **logTrimmedInput**(input) {  *// Initialize startIndex at the beginning of the userInput string.*  **let** startIndex = 0;    *// Initialize endIndex at the end of the userInput string.*  **let** endIndex = userInput.length - 1;   *// Increment startIndex as long as the character at the current index of userInput is a space.*  *// This loop will stop when it finds the first non-space character or reaches the end of the string.*  **while** (userInput[startIndex] === ' ') {  startIndex++;  }   *// Decrement endIndex as long as the character at the current index of userInput is a space.*  *// This loop will stop when it finds the last non-space character or reaches the beginning of the string.*  **while** (userInput[endIndex] === ' ') {  endIndex--;  }   *// Extract the substring from startIndex to endIndex using the slice method.*  *// Since slice does not include the character at the endIndex, we add 1 to include it.*  **const** trimmedInput = userInput.slice(startIndex, endIndex + 1);    *// Return the trimmed userInput string.*  **return** trimmedInput; }  *// Call the logTrimmedInput function with the userInput string and store the result.* **const** trimmedInput = logTrimmedInput(userInput);  *// Display the trimmed userInput string in the console.* console.log(trimmedInput); |

#### **[Exercise 3: Validate Non-empty Input](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-validate-non-empty-input)**

Problem Statement:  
Given a user input string, validate that it is non-empty after trimming using the trim() method.

const userInput = " ";

function isValidInput(input) {

// Use trim() to validate non-empty input

}

const isValid = isValidInput(userInput);

console.log(isValid);

Expected Output:  
false

Solution:

// Sample data: A user input string

const userInput = " ";

// Function to validate non-empty input after trimming

function isValidInput(input) {

// Using trim() to remove whitespace from both ends of the input

// Check if the trimmed input is not an empty string

return input.trim() !== '';

}

// Validate the user input

const isValid = isValidInput(userInput);

console.log(isValid); // Expected Output: false

Alternative solution using loop:

// Sample data: A user input string

const userInput = " ";

// Function to validate non-empty input without using trim()

function isValidInput(input) {

// Loop through each character in the input

for (let i = 0; i < input.length; i++) {

// Check if the current character is not a whitespace character

if (input[i] !== ' ' && input[i] !== '\n' && input[i] !== '\t' && input[i] !== '\r') {

return true; // Found a non-whitespace character, so input is valid

}

}

return false; // No non-whitespace character found, so input is invalid

}

// Validate the user input

const isValid = isValidInput(userInput);

console.log(isValid); // Expected Output: false

# [toUpperCase()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "touppercase)

#### **[Exercise 1: Convert to Uppercase](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-convert-to-uppercase)**

Problem Statement:  
Given a string, convert it to uppercase using the toUpperCase() method.

const string = "javascript";

function convertToUppercase(string) {

// Use toUpperCase() to convert the string

}

const uppercasedString = convertToUppercase(string);

console.log(uppercasedString);

Expected Output:  
JAVASCRIPT

Solution:

|  |
| --- |
| *// Declare a constant named 'string' and assign the value 'javascript' to it.* **const** string = 'javascript';  *// Define a function named 'convertToUppercase' that takes a single argument 'string'.* **function** **convertToUppercase**(string) {  *// Use the built-in toUpperCase() method to convert the entire input string to uppercase.*  *// Return the uppercase version of the string.*  **return** string.toUpperCase(); }  *// Call the 'convertToUppercase' function with the 'string' constant as an argument.* *// Store the result, which will be the uppercase version of the string, in the 'uppercasedString' constant.* **const** uppercasedString = convertToUppercase(string);  *// Output the 'uppercasedString' to the console.* console.log(uppercasedString); *// Expected Output: JAVASCRIPT* |

Alternative solution using loop and ASCII

// Initialize a constant 'string' with the value 'javascript'.

const string = 'javascript'

// Define a function named 'convertToUppercase' that takes a single argument 'str'.

function convertToUppercase(str) {

// Create an empty string 'convertedText' to store the converted uppercase characters.

let convertedText = ''

// Loop through each character of the input string 'str'.

for (let i = 0; i < str.length; i++) {

// Get the ASCII value of the current character using charCodeAt() method.

const charCode = str.charCodeAt(i)

// Check if the ASCII value of the current character is within the range of lowercase letters.

// ASCII values for 'a' is 97 and for 'z' is 122.

if (charCode >= 97 && charCode <= 122) {

// If the current character is a lowercase letter, subtract 32 from its ASCII value.

// This is because the ASCII difference between uppercase and lowercase letters is 32.

// Then convert this new ASCII value back to a character using String.fromCharCode() method.

convertedText += String.fromCharCode(charCode - 32)

} else {

// If the current character is not a lowercase letter, append it as is to 'convertedText'.

convertedText += str[i]

}

}

// After processing all characters, return the 'convertedText' string.

return convertedText

}

// Call the 'convertToUppercase' function with the initial 'string' and store the result in 'uppercasedString'.

const uppercasedString = convertToUppercase(string)

// Output the 'uppercasedString' to the console.

console.log(uppercasedString) // Outputs: JAVASCRIPT

#### **[Exercise 2: Uppercase First Letter](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-uppercase-first-letter)**

Problem Statement:  
Given a string, convert the first letter to uppercase using the toUpperCase() method.

const string = "javascript";

function uppercaseFirstLetter(string) {

// Use toUpperCase() on the first letter

}

const capitalizedString = uppercaseFirstLetter(string);

console.log(capitalizedString);

Expected Output:  
Javascript

Solution:

// Sample data: A string

const string = "javascript";

// Function to uppercase the first letter of the string

function uppercaseFirstLetter(string) {

// Check if the string is not empty

if (string) {

// Convert the first letter to uppercase and concatenate with the rest of the string

return string.charAt(0).toUpperCase() + string.slice(1);

}

return string;

}

// Convert the first letter to uppercase

const capitalizedString = uppercaseFirstLetter(string);

console.log(capitalizedString); // Expected Output: Javascript

Alternative solution using loop:

// Sample data: A string

const string = "javascript";

// Function to uppercase the first letter of the string using a loop

function uppercaseFirstLetter(string) {

let result = "";

// Loop through each character in the string

for (let i = 0; i < string.length; i++) {

// Check if it's the first character

if (i === 0) {

// Convert the first character to uppercase

result += string[i].toUpperCase();

} else {

// Add the rest of the characters as they are

result += string[i];

}

}

return result;

}

// Convert the first letter to uppercase

const capitalizedString = uppercaseFirstLetter(string);

console.log(capitalizedString); // Expected Output: Javascript

#### **[Exercise 3: Uppercase Log Level](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-uppercase-log-level)**

Problem Statement:  
Given a log level string, log it in uppercase using the toUpperCase() method.

const logLevel = "info";

function logUppercaseLevel(level) {

// Use toUpperCase() and log the level

}

logUppercaseLevel(logLevel);

Expected Output:  
INFO

Solution:

// Sample data: A log level string

const logLevel = "info";

// Function to log the level in uppercase

function logUppercaseLevel(level) {

// Convert the level to uppercase

const upperCaseLevel = level.toUpperCase();

// Log the uppercase level

console.log(upperCaseLevel);

}

// Log the level in uppercase

logUppercaseLevel(logLevel); // Expected Output: INFO

Alternative solution using loop:

// Sample data: A log level string

const logLevel = "info";

// Function to log the level in uppercase without using toUpperCase()

function logUppercaseLevel(level) {

let upperCaseLevel = "";

// Loop through each character in the level string

for (let i = 0; i < level.length; i++) {

// Get the ASCII code of the current character

const charCode = level.charCodeAt(i);

// Check if the character is a lowercase letter

if (charCode >= 97 && charCode <= 122) {

// Convert to uppercase by subtracting 32 from the ASCII code

upperCaseLevel += String.fromCharCode(charCode - 32);

} else {

// If it's not a lowercase letter, add the character as is

upperCaseLevel += level[i];

}

}

// Log the uppercase level

console.log(upperCaseLevel);

}

// Log the level in uppercase using the alternative method

logUppercaseLevel(logLevel); // Expected Output: INFO

# [toLowerCase()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "tolowercase)

#### **[Exercise 1: Convert to Lowercase](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-convert-to-lowercase)**

Problem Statement:  
Given a string, convert it to lowercase using the toLowerCase() method.

const string = "JAVASCRIPT";

function convertToLowercase(string) {

// Use toLowerCase() to convert the string

}

const lowercasedString = convertToLowercase(string);

console.log(lowercasedString);

Expected Output:  
javascript

Solution:

// Sample data: A string

const string = "JAVASCRIPT";

// Function to convert the string to lowercase

function convertToLowercase(string) {

// Converting the string to lowercase

return string.toLowerCase();

}

// Convert the string to lowercase

const lowercasedString = convertToLowercase(string);

console.log(lowercasedString); // Expected Output: javascript

Alternative solution using loop:

// Sample data: A string

const string = "JAVASCRIPT";

// Function to convert the string to lowercase manually

function convertToLowercase(string) {

let lowercasedString = "";

// Loop through each character in the string

for (let i = 0; i < string.length; i++) {

// Get the ASCII code of the current character

const charCode = string.charCodeAt(i);

// Check if the character is an uppercase letter

if (charCode >= 65 && charCode <= 90) {

// Convert to lowercase by adding 32 to the ASCII code

lowercasedString += String.fromCharCode(charCode + 32);

} else {

// If it's not an uppercase letter, add the character as is

lowercasedString += string[i];

}

}

return lowercasedString;

}

// Convert the string to lowercase using the alternative method

const lowercasedString = convertToLowercase(string);

console.log(lowercasedString); // Expected Output: javascript

#### **[Exercise 2: Lowercase Email Address](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-lowercase-email-address)**

Problem Statement:  
Given an email address, convert it to lowercase to standardize it using the toLowerCase() method.

const email = "John.Doe@Example.com";

function lowercaseEmail(email) {

// Use toLowerCase() to convert the email address

}

const standardizedEmail = lowercaseEmail(email);

console.log(standardizedEmail);

Expected Output:  
john.doe@example.com

Solution:

// Sample data: An email address

const email = "John.Doe@Example.com";

// Function to convert the email address to lowercase

function lowercaseEmail(email) {

// Converting the email address to lowercase

return email.toLowerCase();

}

// Convert the email address to lowercase for standardization

const standardizedEmail = lowercaseEmail(email);

console.log(standardizedEmail); // Expected Output: john.doe@example.com

Alternative solution using loop:

// Sample data: An email address

const email = "John.Doe@Example.com";

// Function to convert the email address to lowercase manually

function lowercaseEmail(email) {

let lowercasedEmail = "";

// Loop through each character in the email

for (let i = 0; i < email.length; i++) {

// Get the ASCII code of the current character

const charCode = email.charCodeAt(i);

// Check if the character is an uppercase letter

if (charCode >= 65 && charCode <= 90) {

// Convert to lowercase by adding 32 to the ASCII code

lowercasedEmail += String.fromCharCode(charCode + 32);

} else {

// If it's not an uppercase letter, add the character as is

lowercasedEmail += email[i];

}

}

return lowercasedEmail;

}

// Convert the email address to lowercase using the alternative method

const standardizedEmail = lowercaseEmail(email);

console.log(standardizedEmail); // Expected Output: john.doe@example.com

#### **[Exercise 3: Lowercase User Input](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-lowercase-user-input)**

Problem Statement:  
Given a user input string, convert it to lowercase to handle case-insensitive comparison using the toLowerCase() method.

const userInput = "Yes";

function lowercaseInput(input) {

// Use toLowerCase() to convert the user input

}

const standardizedInput = lowercaseInput(userInput);

console.log(standardizedInput);

Expected Output:  
yes

Solution:

// Sample data: A user input string

const userInput = "Yes";

// Function to convert the user input to lowercase

function lowercaseInput(input) {

// Converting the user input to lowercase

return input.toLowerCase();

}

// Convert the user input to lowercase for standardization

const standardizedInput = lowercaseInput(userInput);

console.log(standardizedInput); // Expected Output: yes

Alternative solution using loop:

// Sample data: A user input string

const userInput = "Yes";

// Function to convert the user input to lowercase manually

function lowercaseInput(input) {

let lowercasedInput = "";

// Loop through each character in the input string

for (let i = 0; i < input.length; i++) {

// Get the ASCII code of the current character

const charCode = input.charCodeAt(i);

// Check if the character is an uppercase letter

if (charCode >= 65 && charCode <= 90) {

// Convert to lowercase by adding 32 to the ASCII code

lowercasedInput += String.fromCharCode(charCode + 32);

} else {

// If it's not an uppercase letter, add the character as is

lowercasedInput += input[i];

}

}

return lowercasedInput;

}

// Convert the user input to lowercase using the alternative method

const standardizedInput = lowercaseInput(userInput);

console.log(standardizedInput); // Expected Output: yes

Object.keys()

Exercise 1: Count Object Properties

**Problem Statement**:

Given a user object, count the number of properties it has using Object.keys().

const user = {

name: "John Doe",

age: 25,

country: "USA",

isActive: true

};

**Code Template**:

function countProperties(obj) {

// Use Object.keys() to count the properties of the object

}

const propertyCount = countProperties(user);

console.log(propertyCount);

**Expected Output**:

4

Exercise 2: Check for Empty Object

**Problem Statement**:

Given an object, check if it has no properties using Object.keys().

const obj = {};

**Code Template**:

function isEmpty(obj) {

// Use Object.keys() to check if the object is empty

}

const emptyCheck = isEmpty(obj);

console.log(emptyCheck);

**Expected Output**:

true

Exercise 3: Extract Object Property Names

**Problem Statement**:

Given a product object, extract all its property names using Object.keys().

const product = {

id: 101,

name: "Laptop",

price: 1000,

category: "Electronics"

};

**Code Template**:

function getPropertyNames(obj) {

// Use Object.keys() to extract property names of the object

}

const properties = getPropertyNames(product);

console.log(properties);

**Expected Output**:

css

["id", "name", "price", "category"]

Exercise 4: Check for Specific Properties

**Problem Statement**:

Given a student object, check if it has properties "name", "age", and "grade" using Object.keys().

const student = {

name: "Alice",

age: 20,

grade: "A"

};

**Code Template**:

function hasRequiredProperties(obj) {

// Use Object.keys() to check if the object has specific properties

}

const hasProperties = hasRequiredProperties(student);

console.log(hasProperties);

**Expected Output**:

true

Object Values

Exercise 1: Calculate Total Order Value

**Problem Statement**:

Given an order object with product prices, calculate the total order value using Object.values().

const order = {

productA: 100,

productB: 200,

productC: 50

};

**Code Template**:

function calculateTotalValue(obj) {

// Use Object.values() to calculate the total order value

}

const totalValue = calculateTotalValue(order);

console.log(totalValue);

**Expected Output**:

350

Exercise 2: Check for Specific Value

**Problem Statement**:

Given a user object, check if any of its properties have a value of "admin" using Object.values().

const user = {

username: "john123",

role: "admin"

};

**Code Template**:

function hasAdminRole(obj) {

// Use Object.values() to check if the object has a value of "admin"

}

const isAdmin = hasAdminRole(user);

console.log(isAdmin);

**Expected Output**:

true

Exercise 3: Count Number of True Values

**Problem Statement**:

Given a settings object with boolean values, count the number of true values using Object.values().

const settings = {

darkMode: true,

notifications: true,

location: false,

analytics: true

};

**Code Template**:

function countTrueValues(obj) {

// Use Object.values() to count the number of true values

}

const trueCount = countTrueValues(settings);

console.log(trueCount);

**Expected Output**:

3

Exercise 4: Check for All Positive Values

**Problem Statement**:

Given a scores object, check if all its values are positive using Object.values().

const scores = {

math: 90,

english: 85,

history: 88

};

**Code Template**:

function areAllValuesPositive(obj) {

// Use Object.values() to check if all values are positive

}

const allPositive = areAllValuesPositive(scores);

console.log(allPositive);

**Expected Output**:

true

Object Methods

# Object.assign

### **[Exercise 1: Merge Objects](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-merge-objects)**

Problem Statement:  
Given two objects, obj1 and obj2, merge them into a new object mergedObj such that the properties from obj2 overwrite the properties from obj1 if they have the same keys.

const obj1 = {

name: "John",

age: 25

};

const obj2 = {

age: 30,

city: "New York"

};

Code Template:

function mergeObjects(obj1, obj2) {

// Use Object.assign() to merge obj1 and obj2

}

const mergedObj = mergeObjects(obj1, obj2);

console.log(mergedObj);

Expected Output:

{

name: "John",

age: 30,

city: "New York"

}

Solution:

// Define two objects, obj1 and obj2, each with some properties

const obj1 = {

name: 'John',

age: 25,

}

const obj2 = {

age: 30,

city: 'New York',

}

// Define a function named mergeObjects that takes two objects (obj1 and obj2) as arguments

function mergeObjects(obj1, obj2) {

// Create a new empty object named mergedObj that will hold the merged properties of obj1 and obj2

let mergedObj = {}

// Use a for...in loop to iterate through all properties (keys) of obj1

for (let key in obj1) {

// For each property of obj1, copy the property and its value to mergedObj

mergedObj[key] = obj1[key]

}

// Use another for...in loop to iterate through all properties (keys) of obj2

for (let key in obj2) {

// For each property of obj2, copy the property and its value to mergedObj

// If a property of the same name already exists in mergedObj (from obj1), it will be overwritten by the property from obj2

mergedObj[key] = obj2[key]

}

// Return the merged object, which now contains all properties of obj1 and obj2, with obj2's properties overwriting obj1's where they have the same name

return mergedObj

}

// Call mergeObjects, passing obj1 and obj2 as arguments, and store the returned merged object in a constant named mergedObj

const mergedObj = mergeObjects(obj1, obj2)

// Output the merged object to the console to verify the function's behavior

console.log(mergedObj) // Output: { name: "John", age: 30, city: "New York" }

Alternative Solution using loop:

// Define an object named obj1 with properties name and age

const obj1 = {

name: 'John',

age: 25,

}

// Define another object named obj2 with properties age and city

const obj2 = {

age: 30,

city: 'New York',

}

// Define a function named mergeObjects that takes two objects (obj1 and obj2) as arguments

function mergeObjects(obj1, obj2) {

// Use Object.assign to merge obj1 and obj2, with properties from obj2 overwriting those in obj1 if they have the same name

// The merged result is directly assigned to obj1, so obj1 is modified and also returned by Object.assign

return Object.assign(obj1, obj2);

}

// Call mergeObjects, passing obj1 and obj2 as arguments, and store the returned (and now merged) obj1 in a constant named mergedObj

const mergedObj = mergeObjects(obj1, obj2);

// Output the merged object to the console to verify the function's behavior

console.log(mergedObj); // Output: { name: 'John', age: 30, city: 'New York' }

### [Exercise 2: Clone Object](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-11-clone-object)

Problem Statement:  
Create a function that takes an object as an argument and returns a clone of the object.

const user = {

name: "John",

age: 25

};

Code Template:

function cloneObject(obj) {

// Use Object.assign() to clone obj

}

const clonedUser = cloneObject(user);

console.log(clonedUser);

Expected Output:

{

name: "John",

age: 25

}

Solution:

// Sample data: An object

const user = {

name: "John",

age: 25

};

// Function to clone an object using Object.assign()

function cloneObject(obj) {

// Creating a new object and copying properties from obj to the new object

return Object.assign({}, obj);

}

// Clone the user object

const clonedUser = cloneObject(user);

console.log(clonedUser); // Expected Output: { name: "John", age: 25 }

Alternative solution:

// Sample data: An object

const user = {

name: "John",

age: 25

};

// Function to manually clone an object

function cloneObject(obj) {

let clone = {}; // Initialize an empty object for the clone

// Loop through each property in the original object

for (let key in obj) {

// Ensure the property belongs to the object, not its prototype

if (obj.hasOwnProperty(key)) {

// Copy each property to the clone

clone[key] = obj[key];

}

}

return clone; // Return the cloned object

}

// Clone the user object using the manual method

const clonedUser = cloneObject(user);

console.log(clonedUser); // Expected Output: { name: "John", age: 25 }

### [Exercise 3: Merge Multiple Objects](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-12-merge-multiple-objects)

Problem Statement:  
Given three objects, obj1, obj2, and obj3, merge them into a single object. The properties from latter objects should overwrite the previous ones if there are any conflicts.

const obj1 = { a: 1, b: 2 };

const obj2 = { b: 3, c: 4 };

const obj3 = { c: 5, d: 6 };

Code Template:

function mergeMultipleObjects(obj1, obj2, obj3) {

// Use Object.assign() to merge obj1, obj2, and obj3

}

const mergedObject = mergeMultipleObjects(obj1, obj2, obj3);

console.log(mergedObject);

Expected Output:

{

a: 1,

b: 3,

c: 5,

d: 6

}

Solution:

// Sample data: Three objects

const obj1 = { a: 1, b: 2 };

const obj2 = { b: 3, c: 4 };

const obj3 = { c: 5, d: 6 };

// Function to merge obj1, obj2, and obj3

function mergeMultipleObjects(obj1, obj2, obj3) {

// Merging obj1, obj2, and obj3 into a new object

return Object.assign({}, obj1, obj2, obj3);

}

// Merge the objects

const mergedObject = mergeMultipleObjects(obj1, obj2, obj3);

console.log(mergedObject); // Expected Output: { a: 1, b: 3, c: 5, d: 6 }

Alternative solution:

// Sample data: Three objects

const obj1 = { a: 1, b: 2 };

const obj2 = { b: 3, c: 4 };

const obj3 = { c: 5, d: 6 };

// Function to merge obj1, obj2, and obj3 manually

function mergeMultipleObjects(obj1, obj2, obj3) {

let mergedObject = {}; // Initialize an empty object for the merged result

// Function to copy properties from one object to another

function copyProperties(source, target) {

for (let key in source) {

if (source.hasOwnProperty(key)) {

target[key] = source[key];

}

}

}

// Copy properties from each object to the mergedObject

copyProperties(obj1, mergedObject);

copyProperties(obj2, mergedObject);

copyProperties(obj3, mergedObject);

return mergedObject;

}

// Merge the objects

const mergedObject = mergeMultipleObjects(obj1, obj2, obj3);

console.log(mergedObject); // Expected Output: { a: 1, b: 3, c: 5, d: 6 }

## Object.entries

### **[Exercise 1: Convert Object to Array of Key-Value Pairs](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-convert-object-to-array-of-key-value-pairs)**

Problem Statement:  
Given an object user, convert it into an array of key-value pairs.

const user = {

name: "Alice",

age: 30

};

Code Template:

function convertToObjectArray(user) {

// Use Object.entries() to convert user to an array of key-value pairs

}

const objectArray = convertToObjectArray(user);

console.log(objectArray);

Expected Output:

[

["name", "Alice"],

["age", 30]

]

Solution:  
// Define the user object with properties name and age

const user = {

name: "Alice",

age: 30

};

// Define a function named convertToObjectArray that takes a single argument user

function convertToObjectArray(user) {

// Use Object.entries to convert the user object to an array of key-value pairs

// Object.entries returns an array where each element is an array containing a property key and its corresponding value

return Object.entries(user);

}

// Call convertToObjectArray, passing the user object as an argument,

// and store the returned array of key-value pairs in a constant named objectArray

const objectArray = convertToObjectArray(user);

// Output the array of key-value pairs to the console to verify the function's behavior

console.log(objectArray); // Expected Output: [["name", "Alice"], ["age", 30]]

Alternative solution using loop:  
// Define the user object with properties name and age

const user = {

name: "Alice",

age: 30

};

// Define a function named convertToObjectArray that takes a single argument user

function convertToObjectArray(user) {

// Initialize an empty array to hold the key-value pairs

let objectArray = [];

// Use a for...in loop to iterate through each property of the user object

for (let key in user) {

// For each property, create an array containing the property key and its value,

// and push this array into objectArray

objectArray.push([key, user[key]]);

}

// Return the array of key-value pairs

return objectArray;

}

// Call convertToObjectArray, passing the user object as an argument,

// and store the returned array of key-value pairs in a constant named objectArray

const objectArray = convertToObjectArray(user);

// Output the array of key-value pairs to the console to verify the function's behavior

console.log(objectArray); // Expected Output: [["name", "Alice"], ["age", 30]]

### [Exercise 2: Count Object Properties](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-21-count-object-properties)

Problem Statement:  
Create a function that takes an object as an argument and returns the count of properties it has.

const user = {

name: "John",

age: 25,

city: "New York"

};

Code Template:

function countProperties(obj) {

// Use Object.entries() to count the properties of obj

}

const propertyCount = countProperties(user);

console.log(propertyCount);

Expected Output:

3

Solution:

// Sample data: An object

const user = {

name: "John",

age: 25,

city: "New York"

};

// Function to count the properties of an object

function countProperties(obj) {

// Using Object.entries() to get an array of the object's property pairs

// The length of this array represents the count of properties

return Object.entries(obj).length;

}

// Count the properties of the object

const propertyCount = countProperties(user);

console.log(propertyCount); // Expected Output: 3

Alternative solution:

// Sample data: An object

const user = {

name: "John",

age: 25,

city: "New York"

};

// Function to count the properties of an object using a loop

function countProperties(obj) {

let count = 0; // Initialize a counter for the properties

// Loop through each property in the object

for (let key in obj) {

// Ensure the property is directly on the object, not its prototype

if (obj.hasOwnProperty(key)) {

count++; // Increment the count for each property

}

}

return count; // Return the total count of properties

}

// Count the properties of the object

const propertyCount = countProperties(user);

console.log(propertyCount); // Expected Output: 3

### [Exercise 3: Convert Object to Map](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-22-convert-object-to-map)

Problem Statement:  
Create a function that takes an object representing a collection of products and their prices, and returns an array of strings, each string describing a product and its price. The description should be in the format "Product: [product], Price: $[price]".

Code Template:

const products = {

"Apple": 1.25,

"Milk": 0.99,

"Bread": 2.50

};

function describeProducts(productObj) {

// Use Object.entries() to convert productObj to an array of product descriptions

}

const productDescriptions = describeProducts(products);

console.log(productDescriptions);

Expected Output:

[

"Product: Apple, Price: $1.25",

"Product: Milk, Price: $0.99",

"Product: Bread, Price: $2.50"

]

Solution:

// Sample data: An object representing products and their prices

const products = {

Apple: 1.25,

Milk: 0.99,

Bread: 2.5,

}

// Function to describe products and their prices

function describeProducts(productObj) {

// Using Object.entries() to iterate over the product-price pairs

return Object.entries(productObj).map(([product, price]) => {

// For each [product, price] pair, return a descriptive string

return `Product: ${product}, Price: $${price}`

})

}

// Call the 'describeProducts' function with 'products' as an argument and

// store the result in a new constant named 'productDescriptions'

const productDescriptions = describeProducts(products)

// Output the 'productDescriptions' to the console

console.log(productDescriptions)

Alternative solution:

// Sample data: An object representing products and their prices

const products = {

"Apple": 1.25,

"Milk": 0.99,

"Bread": 2.50

};

// Function to manually create descriptive strings for each product

function describeProducts(productObj) {

let descriptions = []; // Initialize an array to hold the descriptions

// Iterate over the object's properties using a for...in loop

for (let product in productObj) {

if (productObj.hasOwnProperty(product)) {

// Construct a description string for each product and add it to the array

descriptions.push(`Product: ${product}, Price: $${productObj[product]}`);

}

}

return descriptions;

}

// Call the 'describeProducts' function with 'products' as an argument and

// store the result in a new constant named 'productDescriptions'

const productDescriptions = describeProducts(products);

// Output the 'productDescriptions' to the console

console.log(productDescriptions);

## Object.keys

### [Exercise 1: Get All Property Names](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-get-all-property-names)

Problem Statement:  
Given an object person, return an array containing all property names of the object.

const person = {

firstName: "Bob",

lastName: "Smith",

age: 40

};

Code Template:

function getPropertyNames(person) {

// Use Object.keys() to get all property names of person

}

const propertyNames = getPropertyNames(person);

console.log(propertyNames);

Expected Output:

["firstName", "lastName", "age"]

Solution:

// Define the person object with properties firstName, lastName, and age

const person = {

firstName: "Bob",

lastName: "Smith",

age: 40

};

// Define a function named getPropertyNames that takes a single argument person

function getPropertyNames(person) {

// Use Object.keys to extract all property names from the person object and return them as an array

// Object.keys returns an array containing the names of all the object's own enumerable string-keyed properties

return Object.keys(person);

}

// Call getPropertyNames, passing the person object as an argument,

// and store the returned array of property names in a constant named propertyNames

const propertyNames = getPropertyNames(person);

// Output the array of property names to the console to verify the function's behavior

console.log(propertyNames); // Expected Output: ["firstName", "lastName", "age"]

Alternative solution using loop:

// Define the person object with properties firstName, lastName, and age

const person = {

firstName: "Bob",

lastName: "Smith",

age: 40

};

// Define a function named getPropertyNames that takes a single argument person

function getPropertyNames(person) {

// Initialize an empty array to hold the property names

let propertyNames = [];

// Use a for...in loop to iterate through each property of the person object

for (let key in person) {

// For each property, push the property name (key) into the propertyNames array

propertyNames.push(key);

}

// Return the array of property names

return propertyNames;

}

// Call getPropertyNames, passing the person object as an argument,

// and store the returned array of property names in a constant named propertyNames

const propertyNames = getPropertyNames(person);

// Output the array of property names to the console to verify the function's behavior

console.log(propertyNames); // Expected Output: ["firstName", "lastName", "age"]

### [Exercise 2: Check Property Existence](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-31-check-property-existence)

Problem Statement:  
Create a function that takes an object and a property key as arguments and returns a boolean indicating whether the property exists in the object.

const user = {

name: "John",

age: 25

};

Code Template:

function hasProperty(obj, key) {

// Use Object.keys() to check if key exists in obj

}

const hasAgeProperty = hasProperty(user, 'age');

console.log(hasAgeProperty);

Expected Output:

true

Solution:

// Sample data: An object

const user = {

name: "John",

age: 25

};

// Function to check if a property key exists in an object

function hasProperty(obj, key) {

// Get an array of the object's keys and check if 'key' is included

return Object.keys(obj).includes(key);

}

// Call the 'hasProperty' function with 'user' and 'age' as arguments and

// store the result in a new constant named 'hasAgeProperty'

const hasAgeProperty = hasProperty(user, 'age');

// Output the 'hasAgeProperty' to the console

console.log(hasAgeProperty); // Expected Output: true

Alternative solution:

// Sample data: An object

const user = {

name: "John",

age: 25

};

// Function to check if a property key exists in an object using a loop

function hasProperty(obj, key) {

// Iterate over the object's keys using a for...in loop

for (let k in obj) {

// Check if the current key matches the provided key

if (k === key) {

return true; // The key exists in the object

}

}

return false; // The key was not found in the object

}

// Call the 'hasProperty' function with 'user' and 'age' as arguments and

// store the result in a new constant named 'hasAgeProperty'

const hasAgeProperty = hasProperty(user, 'age');

// Output the 'hasAgeProperty' to the console

console.log(hasAgeProperty); // Expected Output: true

### [Exercise 3: Get Property Values](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-32-get-property-values)

Problem Statement:  
Create a function that takes an object as an argument and returns an array of all the values of the object.

const user = {

name: "John",

age: 25

};

Code Template:

function getPropertyValues(obj) {

// Use Object.keys() to get all property values of obj

}

const propertyValues = getPropertyValues(user);

console.log(propertyValues);

Expected Output:

["John", 25]

Solution:

// Sample data: An object

const user = {

name: "John",

age: 25

};

// Function to get all property values of an object

function getPropertyValues(obj) {

// Use Object.keys() to get the keys, and map them to their values

return Object.keys(obj).map(key => obj[key]);

}

// Call the 'getPropertyValues' function with 'user' as an argument and

// store the result in a new constant named 'propertyValues'

const propertyValues = getPropertyValues(user);

// Output the 'propertyValues' to the console

console.log(propertyValues); // Expected Output: ["John", 25]

Alternative solution:

// Sample data: An object

const user = {

name: "John",

age: 25

};

// Function to get all property values of an object using a loop

function getPropertyValues(obj) {

let values = []; // Initialize an array to store the property values

// Iterate over the object's keys using a for...in loop

for (let key in obj) {

// Ensure the key belongs to the object itself, not its prototype

if (obj.hasOwnProperty(key)) {

// Add the value corresponding to the key to the values array

values.push(obj[key]);

}

}

return values; // Return the array of values

}

// Call the 'getPropertyValues' function with 'user' as an argument and

// store the result in a new constant named 'propertyValues'

const propertyValues = getPropertyValues(user);

// Output the 'propertyValues' to the console

console.log(propertyValues); // Expected Output: ["John", 25]

Date and Time

## Exercise 1: Calculate Age from Date of Birth

**Problem Statement**:

Given a date of birth as a string, calculate the age of a person as of today.

const dob = "1995-05-15";

**Code Template**:

function calculateAge(dob) {

// Use Date object to calculate age from date of birth

}

const age = calculateAge(dob);

console.log(age);

**Expected Output**:

28 // This will vary depending on the current year

Solution:

// Define a constant named dob with a date of birth as a string

const dob = '1995-05-15'

// Define a function named calculateAge that takes a single argument dob

function calculateAge(dob) {

// Create a new Date object from the dob string

const birthDate = new Date(dob)

// Get the current date and time

const today = new Date()

// Calculate the difference in years between today and the birthDate

let age = today.getFullYear() - birthDate.getFullYear()

// Check if the birthDate has a month and day that comes after today's month and day

// If so, subtract 1 from the age since the birthday hasn't occurred this year yet

const monthDifference = today.getMonth() - birthDate.getMonth()

if (

monthDifference < 0 ||

(monthDifference === 0 && today.getDate() < birthDate.getDate())

) {

age--

}

// Return the calculated age

return age

}

// Call calculateAge, passing the dob constant as an argument,

// and store the returned age value in a constant named age

const age = calculateAge(dob)

// Output the age to the console to verify the function's behavior

console.log(age) // Output will vary depending on the current year; e.g., 28

## Exercise 2: Days Until Next Birthday

**Problem Statement**:

Given a date of birth as a string, calculate the number of days until the person's next birthday.

const dob = "1995-05-15";

**Code Template**:

function daysUntilNextBirthday(dob) {

// Use Date object to calculate days until next birthday

}

const days = daysUntilNextBirthday(dob);

console.log(days);

**Expected Output**:

20 // This will vary depending on the current date

Solution:

function daysUntilNextBirthday(dob) {

// Current date

const currentDate = new Date()

// Date of birth

const birthDate = new Date(dob)

// Adjust the year of the birth date to the next birthday year

birthDate.setFullYear(currentDate.getFullYear())

// If the birthday has already passed this year, move it to the next year

if (birthDate < currentDate) {

birthDate.setFullYear(currentDate.getFullYear() + 1)

}

// Calculate the difference in milliseconds

const difference = birthDate - currentDate

// Convert the difference from milliseconds to days

const days = Math.ceil(difference / (1000 \* 60 \* 60 \* 24))

return days

}

// Sample data: Date of birth

const dob = '1995-05-15'

// Calculate days until next birthday

const days = daysUntilNextBirthday(dob)

// Output the result

console.log(days) // Output will vary depending on the current date

Math

## Exercise 1: Calculate Area of a Circle

**Problem Statement**:

Given the radius of a circle, calculate its area using the Math object.

const radius = 5;

**Code Template**:

function calculateArea(radius) {

// Use Math object to calculate area of the circle

}

const area = calculateArea(radius);

console.log(area);

**Expected Output**:

78.53981633974483

Solution:

// Sample data: Radius of a circle

const radius = 5;

// Function to calculate the area of a circle

function calculateArea(radius) {

// Use the formula πr^2 to calculate the area

return Math.PI \* Math.pow(radius, 2);

}

// Calculate the area of the circle

const area = calculateArea(radius);

// Output the area to the console

console.log(area); // Expected Output: 78.53981633974483

## Exercise 2: Generate a Random Number within a Range

**Problem Statement**:

Generate a random number between a given range min and max.

const min = 10;

const max = 50;

**Code Template**:

function generateRandomNumber(min, max) {

// Use Math object to generate a random number within the range

}

const randomNumber = generateRandomNumber(min, max);

console.log(randomNumber);

**Expected Output**:

35 // This is just an example. The output will vary as it's random.

Solution:

// Sample data: Minimum and maximum range for the random number

const min = 10;

const max = 50;

// Function to generate a random number within a given range

function generateRandomNumber(min, max) {

// Math.random() returns a random number between 0 and 1.

// Multiply by (max - min + 1) to scale it to the desired range length,

// and add min to shift the range to start from min.

return Math.floor(Math.random() \* (max - min + 1)) + min;

}

// Generate a random number within the range

const randomNumber = generateRandomNumber(min, max);

// Output the random number to the console

console.log(randomNumber); // Output will vary as it's random

Number

## Exercise 1: Format Number with Two Decimal Places

**Problem Statement**:

Given a number, format it to have exactly two decimal places.

const num = 123.4567;

**Code Template**:

function formatNumber(num) {

// Use Number object or methods to format the number

}

const formattedNumber = formatNumber(num);

console.log(formattedNumber);

**Expected Output**:

123.46

Solution:

// Sample data: A number

const num = 123.4567;

// Function to format a number to two decimal places

function formatNumber(num) {

// Use toFixed() to format the number to two decimal places

return num.toFixed(2);

}

// Format the number and store the result in 'formattedNumber'

const formattedNumber = formatNumber(num);

// Output the formatted number to the console

console.log(formattedNumber); // Expected Output: 123.46

## Exercise 2: Check if a Value is an Integer

**Problem Statement**:

Given a value, check if it is an integer.

const value = 123.45;

**Code Template**:

function isInteger(value) {

// Use Number object or methods to check if the value is an integer

}

const check = isInteger(value);

console.log(check);

**Expected Output**:

False

Solution:

// Sample data: A value

const value = 123.45;

// Function to check if a value is an integer

function isInteger(value) {

// Use Number.isInteger() to check if the value is an integer

return Number.isInteger(value);

}

// Check if the value is an integer and store the result in 'check'

const check = isInteger(value);

// Output the result to the console

console.log(check); // Expected Output: false

Array Methods

# find()

## Exercise 1: Find First Negative Number

Problem Statement:  
Given an array of numbers, find the first negative number using the find() method.

const numbers = [3, 5, -2, 8, -7, 6];

function findFirstNegative(numbers) {

// Use find() to get the first negative number

}

const firstNegative = findFirstNegative(numbers);

console.log(firstNegative);

Expected Output:  
-2

Solution:

|  |
| --- |
| *// Initialize an array named 'numbers' with both positive and negative values.* **const** numbers = [3, 5, -2, 8, -7, 6];  *// Define a function named 'findFirstNegative' that takes an array 'numbers' as an argument.* **function** **findFirstNegative**(numbers) {  *// Use the find() method on the 'numbers' array.*  *// The find() method will iterate through each element in the 'numbers' array and*   *// apply the provided function to check if the element satisfies a condition.*  **return** numbers.find(**function**(number) {  *// The condition checks if the current 'number' being tested is less than 0, which means it's negative.*  *// If the condition is met, the find() method will immediately return that number and stop further checking.*  **return** number < 0;  }); }  *// Call the 'findFirstNegative' function with the 'numbers' array as an argument and store the result*  *// (which will be the first negative number found, or 'undefined' if no negative number exists)*  *// in the 'firstNegative' constant.* **const** firstNegative = findFirstNegative(numbers);  *// Output the result to the console.* console.log(firstNegative); *// Expected Output: -2* |

Alternative solution with loop

|  |
| --- |
| *// Initialize an array named 'numbers' with both positive and negative values.* **const** numbers = [3, 5, -2, 8, -7, 6];  *// Define a function named 'findFirstNegative' that takes an array 'numbers' as an argument.* **function** **findFirstNegative**(numbers) {  *// Loop through each element in the 'numbers' array.*  **for** (**let** i = 0; i < numbers.length; i++) {  *// Check if the current element is negative (i.e., less than 0).*  **if** (numbers[i] < 0) {  *// If a negative number is found, return it immediately.*  **return** numbers[i];  }  }  *// If no negative number is found after checking all elements, return 'undefined'.*  *// (This is not explicitly required because functions return 'undefined' by default if no value is returned.)* }  *// Call the 'findFirstNegative' function with the 'numbers' array as an argument and store the result*  *// (which will be the first negative number found, or 'undefined' if no negative number exists)*  *// in the 'firstNegative' constant.* **const** firstNegative = findFirstNegative(numbers);  *// Output the result to the console.* console.log(firstNegative); *// Expected Output: -2* |

## Exercise 2: Find First Capitalized Word

Problem Statement:  
Given an array of words, find the first word that is capitalized using the find() method.

const words = ["apple", "Banana", "cherry", "Date"];

function findFirstCapitalizedWord(words) {

// Use find() to get the first capitalized word

}

const capitalizedWord = findFirstCapitalizedWord(words);

console.log(capitalizedWord);

Expected Output:  
Banana

Solution:

// Define an array of words to search for the first capitalized word

const words = ['apple', 'Banana', 'cherry', 'Date']

// Define a function to find the first capitalized word

function findFirstCapitalizedWord(words) {

// Use the find() method to search for the first capitalized word

const capitalizedWord = words.find((word) => {

// Check if the first character of the word is uppercase

return word[0] === word[0].toUpperCase()

})

// Return the first capitalized word found, or undefined if none is found

return capitalizedWord

}

// Call the findFirstCapitalizedWord function with the words array

const capitalizedWord = findFirstCapitalizedWord(words)

// Output the result, which is the first capitalized word found

console.log(capitalizedWord)

Alternative solution using loop:

// Define an array of words to search for the first capitalized word

const words = ['apple', 'Banana', 'cherry', 'Date']

// Define a function to find the first capitalized word using double for loops

function findFirstCapitalizedWord(words) {

// Iterate through each word in the array

for (let i = 0; i < words.length; i++) {

const word = words[i]

// Iterate through each character in the word

for (let j = 0; j < word.length; j++) {

const char = word[j]

// Check if the character is uppercase

if (char === char.toUpperCase()) {

// Return the capitalized word and exit the loops

return word

}

}

}

// If no capitalized word is found, return undefined

return undefined

}

// Call the findFirstCapitalizedWord function with the words array

const capitalizedWord = findFirstCapitalizedWord(words)

// Output the result, which is the first capitalized word found

console.log(capitalizedWord)

## Exercise 3: Find First Prime Number

Problem Statement:  
Given an array of numbers, find the first prime number using the find() method.

const numbers = [4, 6, 8, 11, 13];

function findFirstPrime(numbers) {

// Use find() to get the first prime number

}

const firstPrime = findFirstPrime(numbers);

console.log(firstPrime);

Expected Output:  
11

Solution:

// Sample data: An array of numbers

const numbers = [4, 6, 8, 11, 13];

// Helper function to check if a number is prime

function isPrime(num) {

if (num <= 1) return false; // Check if the number is less than or equal to 1

for (let i = 2; i < num; i++) {

if (num % i === 0) return false; // Check if the number has any divisors other than 1 and itself

}

return true; // The number is prime

}

// Function to find the first prime number in an array

function findFirstPrime(numbers) {

// Use find() with the isPrime function to find the first prime number

return numbers.find(isPrime);

}

// Call the 'findFirstPrime' function with 'numbers' as an argument and

// store the result in a new constant named 'firstPrime'

const firstPrime = findFirstPrime(numbers);

// Output the 'firstPrime' to the console

console.log(firstPrime); // Expected Output: 11

Alternative solution:

// Sample data: An array of numbers

const numbers = [4, 6, 8, 11, 13];

// Helper function to check if a number is prime

function isPrime(num) {

if (num <= 1) return false; // Check if the number is less than or equal to 1

for (let i = 2; i < num; i++) {

if (num % i === 0) return false; // Check if the number has any divisors other than 1 and itself

}

return true; // The number is prime

}

// Function to find the first prime number in an array using a loop

function findFirstPrime(numbers) {

// Iterate through each number in the array

for (let i = 0; i < numbers.length; i++) {

if (isPrime(numbers[i])) {

return numbers[i]; // Return the first prime number found

}

}

return undefined; // Return undefined if no prime number is found

}

// Call the 'findFirstPrime' function with 'numbers' as an argument and

// store the result in a new constant named 'firstPrime'

const firstPrime = findFirstPrime(numbers);

// Output the 'firstPrime' to the console

console.log(firstPrime); // Expected Output: 11

# findIndex()

## Exercise 1: Find Index of First Negative Number

Problem Statement:  
Given an array of numbers, find the index of the first negative number using the findIndex() method.

const numbers = [3, 5, -2, 8, -7, 6];

function findIndexOfFirstNegative(numbers) {

// Use findIndex() to get the index of the first negative number

}

const indexOfFirstNegative = findIndexOfFirstNegative(numbers);

console.log(indexOfFirstNegative);

Expected Output:  
2

Solution:

// Sample data: An array of numbers

const numbers = [3, 5, -2, 8, -7, 6];

// Function to find the index of the first negative number in an array

function findIndexOfFirstNegative(numbers) {

// Use findIndex() to check for the first negative number

return numbers.findIndex(number => number < 0);

}

// Call the 'findIndexOfFirstNegative' function with 'numbers' as an argument and

// store the result in a new constant named 'indexOfFirstNegative'

const indexOfFirstNegative = findIndexOfFirstNegative(numbers);

// Output the 'indexOfFirstNegative' to the console

console.log(indexOfFirstNegative); // Expected Output: 2

Alternative solution:

// Sample data: An array of numbers

const numbers = [3, 5, -2, 8, -7, 6];

// Function to find the index of the first negative number in an array using a loop

function findIndexOfFirstNegative(numbers) {

// Iterate through each number in the array

for (let i = 0; i < numbers.length; i++) {

// Check if the current number is negative

if (numbers[i] < 0) {

return i; // Return the index of the first negative number found

}

}

return -1; // Return -1 if no negative number is found

}

// Call the 'findIndexOfFirstNegative' function with 'numbers' as an argument and

// store the result in a new constant named 'indexOfFirstNegative'

const indexOfFirstNegative = findIndexOfFirstNegative(numbers);

// Output the 'indexOfFirstNegative' to the console

console.log(indexOfFirstNegative); // Expected Output: 2

## Exercise 2: Find Index of First Capitalized Word

Problem Statement:  
Given an array of words, find the index of the first word that is capitalized using the findIndex() method.

const words = ["apple", "Banana", "cherry", "Date"];

function findIndexOfFirstCapitalizedWord(words) {

// Use findIndex() to get the index of the first capitalized word

}

const indexOfCapitalizedWord = findIndexOfFirstCapitalizedWord(words);

console.log(indexOfCapitalizedWord);

Expected Output:  
1

Solution:

// Sample data: An array of words

const words = ["apple", "Banana", "cherry", "Date"];

// Function to find the index of the first capitalized word in an array

function findIndexOfFirstCapitalizedWord(words) {

// Use findIndex() to check for the first word starting with an uppercase letter

return words.findIndex(word => word.charAt(0) === word.charAt(0).toUpperCase());

}

// Call the 'findIndexOfFirstCapitalizedWord' function with 'words' as an argument and

// store the result in a new constant named 'indexOfCapitalizedWord'

const indexOfCapitalizedWord = findIndexOfFirstCapitalizedWord(words);

// Output the 'indexOfCapitalizedWord' to the console

console.log(indexOfCapitalizedWord); // Expected Output: 1

Alternative solution:

// Sample data: An array of words

const words = ["apple", "Banana", "cherry", "Date"];

// Function to find the index of the first capitalized word in an array using a loop

function findIndexOfFirstCapitalizedWord(words) {

// Iterate through each word in the array

for (let i = 0; i < words.length; i++) {

// Check if the first character of the word is uppercase

if (words[i].charAt(0) === words[i].charAt(0).toUpperCase()) {

return i; // Return the index of the first capitalized word found

}

}

return -1; // Return -1 if no capitalized word is found

}

// Call the 'findIndexOfFirstCapitalizedWord' function with 'words' as an argument and

// store the result in a new constant named 'indexOfCapitalizedWord'

const indexOfCapitalizedWord = findIndexOfFirstCapitalizedWord(words);

// Output the 'indexOfCapitalizedWord' to the console

console.log(indexOfCapitalizedWord); // Expected Output: 1

## Exercise 3: Find Index of First Prime Number

Problem Statement:  
Given an array of numbers, find the index of the first prime number using the findIndex() method.

const numbers = [4, 6, 8, 11, 13];

function findIndexOfFirstPrime(numbers) {

// Use findIndex() to get the index of the first prime number

}

const indexOfFirstPrime = findIndexOfFirstPrime(numbers);

console.log(indexOfFirstPrime);

Expected Output:  
3

Solution:

// Sample data: An array of numbers

const numbers = [4, 6, 8, 11, 13];

// Helper function to check if a number is prime

function isPrime(num) {

if (num <= 1) return false; // Numbers less than or equal to 1 are not prime

for (let i = 2; i < num; i++) {

if (num % i === 0) return false; // If divisible by any number other than 1 and itself, not prime

}

return true; // Otherwise, the number is prime

}

// Function to find the index of the first prime number in an array

function findIndexOfFirstPrime(numbers) {

// Use findIndex() with the isPrime helper function

return numbers.findIndex(isPrime);

}

// Call the 'findIndexOfFirstPrime' function with 'numbers' as an argument and

// store the result in a new constant named 'indexOfFirstPrime'

const indexOfFirstPrime = findIndexOfFirstPrime(numbers);

// Output the 'indexOfFirstPrime' to the console

console.log(indexOfFirstPrime); // Expected Output: 3

Alternative solution:

// Sample data: An array of numbers

const numbers = [4, 6, 8, 11, 13];

// Helper function to check if a number is prime

function isPrime(num) {

if (num <= 1) return false; // Numbers less than or equal to 1 are not prime

for (let i = 2; i < num; i++) {

if (num % i === 0) return false; // If divisible by any number other than 1 and itself, not prime

}

return true; // Otherwise, the number is prime

}

// Function to find the index of the first prime number in an array using a loop

function findIndexOfFirstPrime(numbers) {

// Iterate through each number in the array

for (let i = 0; i < numbers.length; i++) {

if (isPrime(numbers[i])) {

return i; // Return the index of the first prime number found

}

}

return -1; // Return -1 if no prime number is found

}

// Call the 'findIndexOfFirstPrime' function with 'numbers' as an argument and

// store the result in a new constant named 'indexOfFirstPrime'

const indexOfFirstPrime = findIndexOfFirstPrime(numbers);

// Output the 'indexOfFirstPrime' to the console

console.log(indexOfFirstPrime); // Expected Output: 3

# [sort()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "sort)

## [Exercise 1: Sort Numbers in Descending Order](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-sort-numbers-in-descending-order)

Problem Statement:  
Given an array of numbers, sort them in descending order using the sort() method.

const numbers = [3, 5, -2, 8, -7, 6];

function sortDescending(numbers) {

// Use sort() to arrange the numbers in descending order

}

const sortedNumbers = sortDescending(numbers);

console.log(sortedNumbers);

Expected Output:  
[8, 6, 5, 3, -2, -7]

Solution:

// Sample data: An array of numbers

const numbers = [3, 5, -2, 8, -7, 6];

// Function to sort the numbers in descending order

function sortDescending(numbers) {

// Using sort() with a comparison function for descending order

return numbers.sort((a, b) => b - a);

}

// Call the 'sortDescending' function with 'numbers' as an argument and

// store the result in a new constant named 'sortedNumbers'

const sortedNumbers = sortDescending(numbers);

// Output the 'sortedNumbers' to the console

console.log(sortedNumbers); // Expected Output: [8, 6, 5, 3, -2, -7]

Alternative solution:

// Sample data: An array of numbers

const numbers = [3, 5, -2, 8, -7, 6];

// Function to sort the numbers in descending order using bubble sort

function sortDescending(numbers) {

let len = numbers.length;

let swapped;

do {

swapped = false;

for (let i = 0; i < len - 1; i++) {

// Compare adjacent elements and swap if in wrong order

if (numbers[i] < numbers[i + 1]) {

// Swap elements

let temp = numbers[i];

numbers[i] = numbers[i + 1];

numbers[i + 1] = temp;

swapped = true;

}

}

} while (swapped);

return numbers;

}

// Call the 'sortDescending' function with 'numbers' as an argument and

// store the result in a new constant named 'sortedNumbers'

const sortedNumbers = sortDescending(numbers);

// Output the 'sortedNumbers' to the console

console.log(sortedNumbers); // Expected Output: [8, 6, 5, 3, -2, -7]

## [Exercise 2: Sort Words Alphabetically](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-sort-words-alphabetically)

Problem Statement:  
Given an array of words, sort them alphabetically using the sort() method.

const words = ["apple", "Banana", "cherry", "Date"];

function sortAlphabetically(words) {

// Use sort() to arrange the words alphabetically

}

const sortedWords = sortAlphabetically(words);

console.log(sortedWords);

Expected Output:  
["Banana", "Date", "apple", "cherry"]

Solution:

// Sample data: An array of words

const words = ["apple", "Banana", "cherry", "Date"];

// Function to sort the words alphabetically

function sortAlphabetically(words) {

// Using sort() for a case-sensitive alphabetical sort

return words.sort();

}

// Call the 'sortAlphabetically' function with 'words' as an argument and

// store the result in a new constant named 'sortedWords'

const sortedWords = sortAlphabetically(words);

// Output the 'sortedWords' to the console

console.log(sortedWords); // Expected Output: ["Banana", "Date", "apple", "cherry"]

Alternative solution:

// Sample data: An array of words

const words = ["apple", "Banana", "cherry", "Date"];

// Function to sort the words alphabetically using insertion sort

function sortAlphabetically(words) {

for (let i = 1; i < words.length; i++) {

let currentWord = words[i];

let j = i - 1;

// Compare 'currentWord' with the sorted part of the array and insert it in the correct position

while (j >= 0 && words[j] > currentWord) {

words[j + 1] = words[j];

j--;

}

words[j + 1] = currentWord;

}

return words;

}

// Call the 'sortAlphabetically' function with 'words' as an argument and

// store the result in a new constant named 'sortedWords'

const sortedWords = sortAlphabetically(words);

// Output the 'sortedWords' to the console

console.log(sortedWords); // Expected Output: ["Banana", "Date", "apple", "cherry"]

## [Exercise 3: Sort Strings by Length](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-sort-strings-by-length)

Problem Statement:  
Given an array of strings, sort them based on their length using the sort() method.

const strings = ["short", "medium", "very long string", "longer"];

function sortByLength(strings) {

// Use sort() to arrange the strings by length

}

const sortedStrings = sortByLength(strings);

console.log(sortedStrings);

Expected Output:  
["short", "medium", "longer", "very long string"]

Solution:

// Sample data: An array of strings

const strings = ["short", "medium", "very long string", "longer"];

// Function to sort the strings by their length

function sortByLength(strings) {

// Using sort() with a comparison function for length

return strings.sort((a, b) => a.length - b.length);

}

// Call the 'sortByLength' function with 'strings' as an argument and

// store the result in a new constant named 'sortedStrings'

const sortedStrings = sortByLength(strings);

// Output the 'sortedStrings' to the console

console.log(sortedStrings); // Expected Output: ["short", "medium", "longer", "very long string"]

Alternative solution:

// Sample data: An array of strings

const strings = ["short", "medium", "very long string", "longer"];

// Function to sort the strings by their length using bubble sort

function sortByLength(strings) {

let n = strings.length;

let swapped;

do {

swapped = false;

for (let i = 0; i < n - 1; i++) {

// Compare adjacent elements by length and swap if necessary

if (strings[i].length > strings[i + 1].length) {

// Swap elements

let temp = strings[i];

strings[i] = strings[i + 1];

strings[i + 1] = temp;

swapped = true;

}

}

} while (swapped);

return strings;

}

// Call the 'sortByLength' function with 'strings' as an argument and

// store the result in a new constant named 'sortedStrings'

const sortedStrings = sortByLength(strings);

// Output the 'sortedStrings' to the console

console.log(sortedStrings); // Expected Output: ["short", "medium", "longer", "very long string"]

# [splice()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "splice)

## [Exercise 1: Remove Negative Numbers](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-remove-negative-numbers)

Problem Statement:  
Given an array of numbers, remove all negative numbers using the splice() method.

const numbers = [3, 5, -2, 8, -7, 6];

function removeNegatives(numbers) {

// Use splice() to remove all negative numbers

}

const positiveNumbers = removeNegatives(numbers);

console.log(positiveNumbers);

Expected Output:  
[3, 5, 8, 6]

Solution:

// Sample data: An array of numbers

const numbers = [3, 5, -2, 8, -7, 6];

// Function to remove all negative numbers from an array

function removeNegatives(numbers) {

// Iterate backwards through the array

for (let i = numbers.length - 1; i >= 0; i--) {

// Check if the current element is negative

if (numbers[i] < 0) {

// Remove the negative element with splice()

numbers.splice(i, 1);

}

}

return numbers;

}

// Call the 'removeNegatives' function with 'numbers' as an argument and

// store the result in a new constant named 'positiveNumbers'

const positiveNumbers = removeNegatives(numbers);

// Output the 'positiveNumbers' to the console

console.log(positiveNumbers); // Expected Output: [3, 5, 8, 6]

Alternative solution:

// Sample data: An array of numbers

const numbers = [3, 5, -2, 8, -7, 6];

// Function to remove all negative numbers from an array

function removeNegatives(numbers) {

let positives = []; // Initialize a new array for positive numbers

// Iterate through the original array

for (let i = 0; i < numbers.length; i++) {

// Check if the current element is positive

if (numbers[i] >= 0) {

// Add positive elements to the new array

positives.push(numbers[i]);

}

}

return positives; // Return the new array containing only positive numbers

}

// Call the 'removeNegatives' function with 'numbers' as an argument and

// store the result in a new constant named 'positiveNumbers'

const positiveNumbers = removeNegatives(numbers);

// Output the 'positiveNumbers' to the console

console.log(positiveNumbers); // Expected Output: [3, 5, 8, 6]

## [Exercise 2: Replace First Word](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-replace-first-word)

Problem Statement:  
Given an array of words, replace the first word with "Hello" using the splice() method.

const words = ["apple", "Banana", "cherry", "Date"];

function replaceFirstWord(words) {

// Use splice() to replace the first word

}

const newWords = replaceFirstWord(words);

console.log(newWords);

Expected Output:  
["Hello", "Banana", "cherry", "Date"]

Solution:

// Sample data: An array of words

const words = ["apple", "Banana", "cherry", "Date"];

// Function to replace the first word in an array with "Hello"

function replaceFirstWord(words) {

// Use splice() to replace the first word

words.splice(0, 1, "Hello");

return words;

}

// Call the 'replaceFirstWord' function with 'words' as an argument and

// store the result in a new constant named 'newWords'

const newWords = replaceFirstWord(words);

// Output the 'newWords' to the console

console.log(newWords); // Expected Output: ["Hello", "Banana", "cherry", "Date"]

Alternative solution:

// Sample data: An array of words

const words = ["apple", "Banana", "cherry", "Date"];

// Function to replace the first word in an array with "Hello"

function replaceFirstWord(words) {

let newWords = ["Hello"]; // Start with an array containing "Hello"

// Iterate through the original array starting from the second element

for (let i = 1; i < words.length; i++) {

// Add the rest of the words to the new array

newWords.push(words[i]);

}

return newWords; // Return the new array with "Hello" as the first word

}

// Call the 'replaceFirstWord' function with 'words' as an argument and

// store the result in a new constant named 'newWords'

const newWords = replaceFirstWord(words);

// Output the 'newWords' to the console

console.log(newWords); // Expected Output: ["Hello", "Banana", "cherry", "Date"]

## Exercise 3: Insert Numbers

Problem Statement:  
Given an array of numbers, insert the numbers 10 and 15 at the second position using the splice() method.

const numbers = [3, 5, 8, 6];

function insertNumbers(numbers) {

// Use splice() to insert 10 and 15 at the second position

}

const newNumbers = insertNumbers(numbers);

console.log(newNumbers);

Expected Output:  
[3, 10, 15, 5, 8, 6]

Solution:

// Sample data: An array of numbers

const numbers = [3, 5, 8, 6];

// Function to insert 10 and 15 at the second position in an array

function insertNumbers(numbers) {

// Use splice() to insert 10 and 15 at the second position

numbers.splice(1, 0, 10, 15);

return numbers;

}

// Call the 'insertNumbers' function with 'numbers' as an argument and

// store the result in a new constant named 'newNumbers'

const newNumbers = insertNumbers(numbers);

// Output the 'newNumbers' to the console

console.log(newNumbers); // Expected Output: [3, 10, 15, 5, 8, 6]

Alternative solution:

// Sample data: An array of numbers

const numbers = [3, 5, 8, 6];

// Function to insert 10 and 15 at the second position in an array

function insertNumbers(numbers) {

let newNumbers = []; // Initialize a new array

// Iterate through the original array and construct the new array

for (let i = 0; i < numbers.length; i++) {

if (i === 1) {

// Insert 10 and 15 at the second position

newNumbers.push(10, 15);

}

// Add the current element from the original array

newNumbers.push(numbers[i]);

}

return newNumbers; // Return the new array with the inserted numbers

}

// Call the 'insertNumbers' function with 'numbers' as an argument and

// store the result in a new constant named 'newNumbers'

const newNumbers = insertNumbers(numbers);

// Output the 'newNumbers' to the console

console.log(newNumbers); // Expected Output: [3, 10, 15, 5, 8, 6]

# [slice()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "slice)

#### **[Exercise 1: Extract Subarray](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-extract-subarray)**

Problem Statement:  
Given an array of numbers, extract a subarray from index 1 to index 3 using the slice() method.

const numbers = [3, 5, 8, 6, 7, 9];

function extractSubarray(numbers) {

// Use slice() to extract the subarray

}

const subarray = extractSubarray(numbers);

console.log(subarray);

Expected Output:  
[5, 8, 6]

Solution:

// Sample data: An array of numbers

const numbers = [3, 5, 8, 6, 7, 9];

// Function to extract a subarray from index 1 to index 3

function extractSubarray(numbers) {

// Use slice() to extract the subarray from index 1 to 3 (end index is exclusive)

return numbers.slice(1, 4);

}

// Call the 'extractSubarray' function with 'numbers' as an argument and

// store the result in a new constant named 'subarray'

const subarray = extractSubarray(numbers);

// Output the 'subarray' to the console

console.log(subarray); // Expected Output: [5, 8, 6]

Alternative solution:

// Sample data: An array of numbers

const numbers = [3, 5, 8, 6, 7, 9];

// Function to manually extract a subarray from index 1 to index 3

function extractSubarray(numbers) {

let subarray = []; // Initialize a new array for the subarray

// Iterate through the original array within the specified range

for (let i = 1; i <= 3; i++) {

// Add the elements within the range to the new array

subarray.push(numbers[i]);

}

return subarray; // Return the new array containing the subarray

}

// Call the 'extractSubarray' function with 'numbers' as an argument and

// store the result in a new constant named 'subarray'

const subarray = extractSubarray(numbers);

// Output the 'subarray' to the console

console.log(subarray); // Expected Output: [5, 8, 6]

#### **[Exercise 2: Extract Domain from Email](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-extract-domain-from-email)**

Problem Statement:  
Given an email address, extract the domain using the slice() method.

const email = "john.doe@example.com";

function getDomain(email) {

// Use slice() to extract the domain from the email

}

const domain = getDomain(email);

console.log(domain);

Expected Output:  
example.com

Solution:

// Sample data: An email address

const email = "john.doe@example.com";

// Function to extract the domain from an email address

function getDomain(email) {

// Find the position of the '@' character

let atIndex = email.indexOf('@');

// Use slice() to extract the domain, starting from the character after '@'

return email.slice(atIndex + 1);

}

// Call the 'getDomain' function with 'email' as an argument and

// store the result in a new constant named 'domain'

const domain = getDomain(email);

// Output the 'domain' to the console

console.log(domain); // Expected Output: example.com

Alternative solution:

// Sample data: An email address

const email = "john.doe@example.com";

// Function to manually extract the domain from an email address

function getDomain(email) {

let domain = "";

let atFound = false; // Flag to indicate if '@' has been found

// Iterate through each character in the email string

for (let i = 0; i < email.length; i++) {

// Once '@' is found, start building the domain string

if (atFound) {

domain += email[i];

}

// Check if current character is '@' and set the flag

if (email[i] === '@') {

atFound = true;

}

}

return domain; // Return the domain string

}

// Call the 'getDomain' function with 'email' as an argument and

// store the result in a new constant named 'domain'

const domain = getDomain(email);

// Output the 'domain' to the console

console.log(domain); // Expected Output: example.com

#### **[Exercise 3: Extract Last 3 Characters](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-extract-last-3-characters)**

Problem Statement:  
Given a string, extract the last 3 characters using the slice() method.

const string = "JavaScript";

function getLast3Chars(string) {

// Use slice() to extract the last 3 characters

}

const last3Chars = getLast3Chars(string);

console.log(last3Chars);

Expected Output:  
ipt

Solution:

// Sample data: A string

const string = "JavaScript";

// Function to extract the last 3 characters of a string

function getLast3Chars(string) {

// Use slice() with a negative index to get the last 3 characters

return string.slice(-3);

}

// Call the 'getLast3Chars' function with 'string' as an argument and

// store the result in a new constant named 'last3Chars'

const last3Chars = getLast3Chars(string);

// Output the 'last3Chars' to the console

console.log(last3Chars); // Expected Output: ipt

Alternative solution:

// Sample data: A string

const string = "JavaScript";

// Function to manually extract the last 3 characters of a string using a loop

function getLast3Chars(str) {

let last3Chars = "";

// Start looping from the length of the string minus 3

for (let i = str.length - 3; i < str.length; i++) {

// Append each of the last three characters to 'last3Chars'

last3Chars += str[i];

}

return last3Chars;

}

// Call the 'getLast3Chars' function with 'string' as an argument and

// store the result in a new constant named 'last3Chars'

const last3Chars = getLast3Chars(string);

// Output the 'last3Chars' to the console

console.log(last3Chars); // Expected Output: ipt

# [concat()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "concat)

#### **[Exercise 1: Merge Two Arrays](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-merge-two-arrays)**

Problem Statement:  
Given two arrays of numbers, merge them into a single array using the concat() method.

const array1 = [3, 5, 8];

const array2 = [7, 9, 2];

function mergeArrays(arr1, arr2) {

// Use concat() to merge the arrays

}

const mergedArray = mergeArrays(array1, array2);

console.log(mergedArray);

Expected Output:  
[3, 5, 8, 7, 9, 2]

Solution:

// Sample data: Two arrays of numbers

const array1 = [3, 5, 8];

const array2 = [7, 9, 2];

// Function to merge two arrays into one

function mergeArrays(arr1, arr2) {

// Use concat() to merge arr1 and arr2 into a new array

return arr1.concat(arr2);

}

// Call the 'mergeArrays' function with 'array1' and 'array2' as arguments and

// store the result in a new constant named 'mergedArray'

const mergedArray = mergeArrays(array1, array2);

// Output the 'mergedArray' to the console

console.log(mergedArray); // Expected Output: [3, 5, 8, 7, 9, 2]

Alternative solution:

// Sample data: Two arrays of numbers

const array1 = [3, 5, 8];

const array2 = [7, 9, 2];

// Function to manually merge two arrays

function mergeArrays(arr1, arr2) {

let mergedArray = [];

// Add all elements of arr1 to mergedArray

for (let i = 0; i < arr1.length; i++) {

mergedArray.push(arr1[i]);

}

// Add all elements of arr2 to mergedArray

for (let i = 0; i < arr2.length; i++) {

mergedArray.push(arr2[i]);

}

return mergedArray;

}

// Call the 'mergeArrays' function with 'array1' and 'array2' as arguments and

// store the result in a new constant named 'mergedArray'

const mergedArray = mergeArrays(array1, array2);

// Output the 'mergedArray' to the console

console.log(mergedArray); // Expected Output: [3, 5, 8, 7, 9, 2]

#### **[Exercise 2: Concatenate Words](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-concatenate-words)**

Problem Statement:  
Given two arrays of words, concatenate them into a single array using the concat() method.

const fruits = ["apple", "banana", "cherry"];

const vegetables = ["carrot", "pea", "potato"];

function concatenateWords(arr1, arr2) {

// Use concat() to concatenate the arrays

}

const foods = concatenateWords(fruits, vegetables);

console.log(foods);

Expected Output:  
["apple", "banana", "cherry", "carrot", "pea", "potato"]

Solution:

// Sample data: Two arrays of words

const fruits = ["apple", "banana", "cherry"];

const vegetables = ["carrot", "pea", "potato"];

// Function to concatenate two arrays of words

function concatenateWords(arr1, arr2) {

// Use concat() to combine arr1 and arr2 into a new array

return arr1.concat(arr2);

}

// Call the 'concatenateWords' function with 'fruits' and 'vegetables' as arguments and

// store the result in a new constant named 'foods'

const foods = concatenateWords(fruits, vegetables);

// Output the 'foods' to the console

console.log(foods); // Expected Output: ["apple", "banana", "cherry", "carrot", "pea", "potato"]

Alternative solution:

// Sample data: Two arrays of words

const fruits = ["apple", "banana", "cherry"];

const vegetables = ["carrot", "pea", "potato"];

// Function to manually concatenate two arrays of words

function concatenateWords(arr1, arr2) {

let combinedArray = [];

// Add all elements of arr1 to combinedArray

for (let i = 0; i < arr1.length; i++) {

combinedArray.push(arr1[i]);

}

// Add all elements of arr2 to combinedArray

for (let i = 0; i < arr2.length; i++) {

combinedArray.push(arr2[i]);

}

return combinedArray;

}

// Call the 'concatenateWords' function with 'fruits' and 'vegetables' as arguments and

// store the result in a new constant named 'foods'

const foods = concatenateWords(fruits, vegetables);

// Output the 'foods' to the console

console.log(foods); // Expected Output: ["apple", "banana", "cherry", "carrot", "pea", "potato"]

#### **[Exercise 3: Concatenate Multiple Arrays](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-concatenate-multiple-arrays)**

Problem Statement:  
Given three arrays of numbers, concatenate them into a single array using the concat() method.

const array1 = [1, 2, 3];

const array2 = [4, 5, 6];

const array3 = [7, 8, 9];

function concatenateMultipleArrays(arr1, arr2, arr3) {

// Use concat() to concatenate the arrays

}

const concatenatedArray = concatenateMultipleArrays(array1, array2, array3);

console.log(concatenatedArray);

Expected Output:  
[1, 2, 3, 4, 5, 6, 7, 8, 9]

Solution:

// Sample data: Three arrays of numbers

const array1 = [1, 2, 3];

const array2 = [4, 5, 6];

const array3 = [7, 8, 9];

// Function to concatenate three arrays

function concatenateMultipleArrays(arr1, arr2, arr3) {

// Use concat() to combine arr1, arr2, and arr3 into a new array

return arr1.concat(arr2, arr3);

}

// Call the 'concatenateMultipleArrays' function with 'array1', 'array2', and 'array3' as arguments and

// store the result in a new constant named 'concatenatedArray'

const concatenatedArray = concatenateMultipleArrays(array1, array2, array3);

// Output the 'concatenatedArray' to the console

console.log(concatenatedArray); // Expected Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Alternative solution:

// Sample data: Three arrays of numbers

const array1 = [1, 2, 3];

const array2 = [4, 5, 6];

const array3 = [7, 8, 9];

// Function to manually concatenate three arrays

function concatenateMultipleArrays(arr1, arr2, arr3) {

let combinedArray = [];

// Add all elements of arr1 to combinedArray

for (let element of arr1) {

combinedArray.push(element);

}

// Add all elements of arr2 to combinedArray

for (let element of arr2) {

combinedArray.push(element);

}

// Add all elements of arr3 to combinedArray

for (let element of arr3) {

combinedArray.push(element);

}

return combinedArray;

}

// Call the 'concatenateMultipleArrays' function with 'array1', 'array2', and 'array3' as arguments and

// store the result in a new constant named 'concatenatedArray'

const concatenatedArray = concatenateMultipleArrays(array1, array2, array3);

// Output the 'concatenatedArray' to the console

console.log(concatenatedArray); // Expected Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

# [join()](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "join)

#### **[Exercise 1: Create a Sentence](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-1-create-a-sentence)**

Problem Statement:  
Given an array of words, create a sentence by joining them using the join() method.

const words = ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"];

function createSentence(words) {

// Use join() to create a sentence

}

const sentence = createSentence(words);

console.log(sentence);

Expected Output:  
The quick brown fox jumps over the lazy dog

Solution:

// Sample data: An array of words

const words = ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"];

// Function to create a sentence from an array of words

function createSentence(words) {

// Use join() with a space as the separator to create a sentence

return words.join(" ");

}

// Call the 'createSentence' function with 'words' as an argument and

// store the result in a new constant named 'sentence'

const sentence = createSentence(words);

// Output the 'sentence' to the console

console.log(sentence); // Expected Output: The quick brown fox jumps over the lazy dog

Alternative solution:

// Sample data: An array of words

const words = ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"];

// Function to create a sentence from an array of words

function createSentence(words) {

let sentence = "";

// Iterate through the words array

for (let i = 0; i < words.length; i++) {

// Concatenate each word to the sentence

sentence += words[i];

// Add a space after each word except the last one

if (i < words.length - 1) {

sentence += " ";

}

}

return sentence;

}

// Call the 'createSentence' function with 'words' as an argument and

// store the result in a new constant named 'sentence'

const sentence = createSentence(words);

// Output the 'sentence' to the console

console.log(sentence); // Expected Output: The quick brown fox jumps over the lazy dog

#### **[Exercise 2: Create a Comma-Separated List](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-2-create-a-comma-separated-list)**

Problem Statement:  
Given an array of fruits, create a comma-separated list using the join() method.

const fruits = ["apple", "banana", "cherry"];

function createList(fruits) {

// Use join() to create a comma-separated list

}

const fruitList = createList(fruits);

console.log(fruitList);

Expected Output:  
apple,banana,cherry

Solution:

// Sample data: An array of fruits

const fruits = ["apple", "banana", "cherry"];

// Function to create a comma-separated list from an array of fruits

function createList(fruits) {

// Use join() with a comma as the separator to create the list

return fruits.join(",");

}

// Call the 'createList' function with 'fruits' as an argument and

// store the result in a new constant named 'fruitList'

const fruitList = createList(fruits);

// Output the 'fruitList' to the console

console.log(fruitList); // Expected Output: apple,banana,cherry

Alternative solution:

// Sample data: An array of fruits

const fruits = ["apple", "banana", "cherry"];

// Function to create a comma-separated list from an array of fruits

function createList(fruits) {

let fruitList = "";

// Iterate through the fruits array

for (let i = 0; i < fruits.length; i++) {

// Concatenate each fruit to the fruitList

fruitList += fruits[i];

// Add a comma after each fruit except the last one

if (i < fruits.length - 1) {

fruitList += ",";

}

}

return fruitList;

}

// Call the 'createList' function with 'fruits' as an argument and

// store the result in a new constant named 'fruitList'

const fruitList = createList(fruits);

// Output the 'fruitList' to the console

console.log(fruitList); // Expected Output: apple,banana,cherry

#### **[Exercise 3: Create a Dashed String](https://github.com/ramGit22/javascript-book/new/master?readme=1" \l "exercise-3-create-a-dashed-string)**

Problem Statement:  
Given an array of characters, create a string by joining them with dashes using the join() method.

const characters = ["J", "a", "v", "a", "S", "c", "r", "i", "p", "t"];

function createDashedString(characters) {

// Use join() to create a dashed string

}

const dashedString = createDashedString(characters);

console.log(dashedString);

Expected Output:  
J-a-v-a-S-c-r-i-p-t

Solution:

// Sample data: An array of characters

const characters = ['J', 'a', 'v', 'a', 'S', 'c', 'r', 'i', 'p', 't']

// Function to create a dashed string from an array of characters

function createDashedString(characters) {

// Use join() with a dash as the separator to create the string

return characters.join('-')

}

// Call the 'createDashedString' function with 'characters' as an argument and

// store the result in a new constant named 'dashedString'

const dashedString = createDashedString(characters)

// Output the 'dashedString' to the console

console.log(dashedString) // Expected Output: J-a-v-a-S-c-r-i-p-t

Alternative solution:

// Sample data: An array of characters

const characters = ['J', 'a', 'v', 'a', 'S', 'c', 'r', 'i', 'p', 't']

// Function to create a dashed string from an array of characters

function createDashedString(characters) {

let dashedString = ''

// Iterate through the characters array

for (let i = 0; i < characters.length; i++) {

// Append each character to the dashedString

dashedString += characters[i]

// Add a dash after each character except the last one

if (i < characters.length - 1) {

dashedString += '-'

}

}

return dashedString

}

// Call the 'createDashedString' function with 'characters' as an argument and

// store the result in a new constant named 'dashedString'

const dashedString = createDashedString(characters)

// Output the 'dashedString' to the console

console.log(dashedString) // Expected Output: J-a-v-a-S-c-r-i-p-t

*That's a great initiative! Including real-world scenarios for commonly used JavaScript methods will be immensely helpful for learners. Here are some commonly used JavaScript methods and concepts that you can consider including in your eBook:*

* *Array Methods:*
  + *filter(): Filter items in an array based on a condition.*
  + *forEach(): Execute a function for each item in an array.*
  + *some(): Check if at least one item in an array meets a condition.*
  + *every(): Check if all items in an array meet a condition.*
  + *find(): Find the first item in an array that meets a condition.*
  + *findIndex(): Find the index of the first item that meets a condition.*
  + *sort(): Sort the items in an array.*
  + *splice(): Add or remove items from an array.*
  + *slice(): Extract a section of an array.*
  + *concat(): Combine two or more arrays.*
  + *join(): Join the elements of an array into a string.*
* *String Methods:*
  + *charAt(): Get a character at a specific index.*
  + *indexOf(): Find the index of a substring.*
  + *lastIndexOf(): Find the last index of a substring.*
  + *slice(): Extract a section of a string.*
  + *split(): Split a string into an array.*
  + *replace(): Replace a substring.*
  + *toUpperCase(): Convert a string to uppercase.*
  + *toLowerCase(): Convert a string to lowercase.*
  + *trim(): Remove whitespace from both ends of a string.*
* *Object Methods and Concepts:*
  + *Object.keys(): Get an array of an object's keys.*
  + *Object.values(): Get an array of an object's values.*
  + *Object.entries(): Get an array of an object's key-value pairs.*
  + *Object.assign(): Copy values from one object to another.*
  + *Destructuring: Extract properties from objects and arrays.*
  + *Spread Operator: Spread elements of arrays and object properties.*
* *Functional Programming Concepts:*
  + *Callback Functions: Functions passed as arguments to other functions.*
  + *Higher-Order Functions: Functions that take other functions as arguments or return functions.*
  + *Closures: Functions that have access to the parent scope.*
  + *Currying: Transforming a function that takes multiple arguments into a series of functions that take one argument.*
* *Asynchronous Programming:*
  + *Promises: A way to handle asynchronous operations.*
  + *async/await: A syntactic sugar for handling promises.*
  + *fetch(): A method to make network requests.*
* *Date and Time:*
  + *Date object: Methods to get and set date and time.*
  + *Timezones, UTC, and locale-specific formatting.*
* *Math and Number:*
  + *Math object: Methods for mathematical operations.*
  + *Number object: Methods and properties related to numbers.*
* *Regular Expressions:*
  + *Pattern matching in strings.*
  + *Methods like match(), search(), replace(), and test().*