9 JavaScript CODE EXERCISES

Learn and Practice JavaScript Code

9 CODING EXERCISES TEST YOUR SKILLS

Advanced Coding Exercise: Building a Dynamic Survey Form

Introduction:

🎉 Unveiling an Advanced JavaScript Coding Challenge: The Dynamic Survey Form 🎉

Elevate your JavaScript skills to the next level with our latest coding exercise. Dive into the world of variables and data types as you build a dynamic survey form from scratch. This challenge will put your knowledge of strings, numbers, booleans, arrays, and objects to the test, all while giving you hands-on experience with DOM manipulation and event handling.

#JavaScript #CodingChallenge #WebDevelopment #Programming #DataTypes #Variables #DOMManipulation #SoftwareEngineering #TechCommunity

Embrace the challenge and let creativity flow! Can't wait to see your dynamic forms in action. Share your progress, insights, and finished projects below! 💡👩‍💻🚀

Objective: Create a dynamic survey form that collects user responses and displays a summary. This exercise will showcase the handling of various data types in JavaScript, including strings, numbers, booleans, arrays, and objects, and demonstrate how to manipulate DOM elements based on user input.

Requirements:

- Use HTML to create the basic structure of the survey form.
- Use CSS for basic styling.
- Implement JavaScript to dynamically update the survey form and display the summary of responses.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title>Dynamic Survey Form</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="surveyForm">
        <h2>Survey Form</h2>
        <label for="name">Name:</label>
        <input type="text" id="name" placeholder="Enter your name">
        <label for="age">Age:</label>
        <input type="number" id="age" placeholder="Enter your age">
        <label>Favorite Color:</label>
        <select id="color">
            <option value="Red">Red</option>
            <option value="Green">Green</option>
            <option value="Blue">Blue</option>
        </select>
        <button onclick="submitForm()">Submit</button>
    </div>
    <div id="summary" style="display:none;">
        <h2>Survey Summary</h2>
        <p id="summaryText"></p>
    </div>
    <script src="script.js"></script>
</body>
```

```
</html>
```

CSS (style.css):

```
body {
    font-family: Arial, sans-serif;
}
#surveyForm {
    margin: 20px;
    padding: 20px;
    border: 1px solid #ddd;
}
#summary {
    margin: 20px;
    padding: 20px;
    border: 1px solid #ddd;
    background-color: #f9f9f9;
}
```

JavaScript (script.js):

```
function submitForm() {
    // Collecting and storing user input
    const name = document.getElementById('name').value;
    const age = document.getElementById('age').value;
```

```
    const color = document.getElementById('color').value;
    // Validating input
    if (!name || !age) {
        alert('Please fill out all fields.');
        return;
    }
    // Creating a response object
    const response = {
        name: name,
        age: parseInt(age, 10), // Ensuring age is stored as a number
        favoriteColor: color,
    };
    // Displaying the summary
    const summaryElement = document.getElementById('summary');
    const summaryText = `
        Name: ${response.name}<br>
        Age: ${response.age}<br>
        Favorite Color: ${response.favoriteColor}
    `;
    document.getElementById('summaryText').innerHTML = summaryText;
    document.getElementById('surveyForm').style.display = 'none';
    summaryElement.style.display = 'block';
}
```

Explanation:

HTML: Defines the structure of the survey form with input fields for name, age, and favorite color. Also includes a hidden summary section.
CSS: Provides basic styling for the form and summary sections.
JavaScript: Handles the form submission, collects user input, validates the input to ensure it's filled out, and then displays a summary of the responses. This involves manipulating variables of different data types (string, number, object) and working with DOM elements.

Advanced Coding Exercise: Custom Calculator Application

Introduction:

🚀 Elevate Your JavaScript Skills: Advanced Operators Challenge! 🚀
Step up your game with our latest #JavaScript challenge - the Custom Calculator Application. This exercise will test your grasp of various operators, including arithmetic, assignment, comparison, logical, and bitwise.
Build a fully functional calculator that not only adds, subtracts, multiplies, and divides but also performs bitwise operations. A perfect blend of #UI, #UX, and #JavaScriptLogic!

#CodingChallenge #WebDevelopment #Programming #JavaScriptOperators #SoftwareEngineering #FrontEndDevelopment #TechCommunity

Embrace this challenge, share your innovative solutions, and let's discuss the power and versatility of JavaScript operators. Looking forward to seeing your creative calculators! 🧮💻🎉

Objective: Create a custom calculator application that performs basic arithmetic operations, bitwise operations, and demonstrates the use of various JavaScript operators, including arithmetic, assignment, comparison, logical, and bitwise operators.

Requirements:

- Use HTML to create the interface for the calculator with buttons for different operations.
- Implement JavaScript to handle the logic of the calculator.
- The calculator should be able to perform addition, subtraction, multiplication, division, modulo operation, bitwise AND, bitwise OR, and bitwise XOR.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title>Custom Calculator</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="calculator">
        <input type="text" id="display" disabled>
        <div id="buttons">
            <button onclick="appendNumber(1)">1</button>
            <button onclick="appendNumber(2)">2</button>
            <!-- Add other number buttons -->
            <button onclick="setOperator('+')">+</button>
            <button onclick="setOperator('-')">-</button>
            <!-- Add other operator buttons -->
            <button onclick="calculate()">=</button>
            <button onclick="clearDisplay()">C</button>
        </div>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
body {
    font-family: Arial, sans-serif;
```

```
}
#calculator {
    width: 300px;
    margin: auto;
    border: 1px solid #ddd;
    padding: 10px;
}
#buttons button {
    width: 70px;
    height: 35px;
    margin: 2px;
}
```

JavaScript (script.js):

```
let currentNumber = '';
let previousNumber = '';
let operation = null;
function appendNumber(number) {
    if (currentNumber.length <= 10) {
        currentNumber += number;
        updateDisplay();
    }
}
function setOperator(operator) {
```

```
    if (currentNumber === '') return;
    if (previousNumber !== '') calculate();
    operation = operator;
    previousNumber = currentNumber;
    currentNumber = '';
}
function calculate() {
    let result;
    const prev = parseFloat(previousNumber);
    const current = parseFloat(currentNumber);
    if (isNaN(prev) || isNaN(current)) return;
    switch (operation) {
        case '+':
            result = prev + current;
            break;
        case '-':
            result = prev - current;
            break;
        case '*':
            result = prev * current;
            break;
        case '/':
            result = prev / current;
            break;
```

```
        case '%':
            result = prev % current;
            break;
        case '&': // Bitwise AND
            result = prev & current;
            break;
        case '|': // Bitwise OR
            result = prev | current;
            break;
        case '^': // Bitwise XOR
            result = prev ^ current;
            break;
        default:
            return;
    }
    currentNumber = result.toString();
    operation = undefined;
    previousNumber = '';
    updateDisplay();
}
function clearDisplay() {
    currentNumber = '';
    previousNumber = '';
    operation = undefined;
```

```
    updateDisplay();
}
function updateDisplay() {
    document.getElementById('display').value = currentNumber;
}
```

Explanation:

HTML: Sets up the structure of the calculator with input fields and buttons for numbers and operations.

CSS: Provides basic styling for the calculator layout and buttons.

JavaScript: Contains the logic for the calculator. It uses variables to keep track of the current and previous numbers and the selected operation. Functions are implemented for handling numbers and operators, performing calculations, and updating the display.

Advanced Coding Exercise: Dynamic Gallery with Filtering

Introduction:

🌟 Advanced JavaScript Challenge: Dynamic Image Gallery 📸

Elevate your JavaScript skills! Our latest challenge lets you build a dynamic image gallery with advanced functions. Dive into higher-order functions, callbacks, and

IIFEs to create a gallery with real-time filtering. Perfect for honing your #JavaScriptSkills and #WebDevelopment expertise!

🚀 Challenge Highlights:

- Dynamic DOM manipulation
- Advanced JavaScript functions
- Real-time image filtering

🔗 Start the challenge: [YourWebsite.com/DynamicGalleryChallenge]

#CodingChallenge #JavaScript #Function #WebDev #FrontEnd #Programming #GalleryApp #TechCommunity #Developers

Join the challenge, share your solutions, and let's explore the beauty of JavaScript functions together. Can't wait to see your creative galleries! 🎨💻👩‍💻👩‍💻

#DynamicGallery #CodeNewbie #DevLife

Objective: Develop a dynamic image gallery application in JavaScript that demonstrates advanced functions, including higher-order functions, callback functions, and immediately invoked function expressions (IIFE). The gallery should allow users to filter images based on categories.

Requirements:

- Use HTML to create the structure of the image gallery.
- Implement JavaScript functions to dynamically populate and filter the gallery.
- Utilize CSS for basic styling.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Dynamic Image Gallery</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="filterButtons"></div>
    <div id="imageGallery"></div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
#filterButtons button {
    margin: 5px;
    padding: 10px;
}
.image {
    width: 100px;
    height: 100px;
```

```
    margin: 5px;
    display: inline-block;
    background-size: cover;
    background-position: center;
}
```

JavaScript (script.js):

```
(function() {
    const images = [
        { src: 'image1.jpg', category: 'nature' },
        { src: 'image2.jpg', category: 'city' },
        // ... more images
    ];
    function createButton(category) {
        const button = document.createElement('button');
        button.innerText = category;
        button.onclick = () => filterImages(category);
        document.getElementById('filterButtons').appendChild(button);
    }
    function createImageElement(image) {
        const div = document.createElement('div');
        div.className = 'image';
        div.style.backgroundImage = `url(${image.src})`;
        return div;
```

```
    }
    function displayImages(imagesToDisplay) {
        const gallery = document.getElementById('imageGallery');
        gallery.innerHTML = '';
        imagesToDisplay.forEach(image =>
gallery.appendChild(createImageElement(image)));
    }
    function filterImages(category) {
        const filteredImages = images.filter(image => image.category === category);
        displayImages(filteredImages);
    }
    // Initialize gallery
    const categories = [...new Set(images.map(image => image.category))];
    categories.forEach(createButton);
    displayImages(images);
})();
```

Explanation:

HTML: Provides the basic layout for the filter buttons and image gallery.

CSS: Styles the filter buttons and image elements.

JavaScript:

An IIFE encapsulates the entire code to avoid polluting the global scope.

The images array contains objects representing each image and its category.

createButton is a function that dynamically creates a button for each category.

createImageElement generates a div with the background image set to the provided image source.

displayImages clears the gallery and appends image elements for each image in the provided array.

filterImages filters the images array based on the selected category and updates the gallery.

On initialization, the script dynamically generates filter buttons for each unique category and displays all images.

Advanced Coding Exercise: Interactive Quiz Application

Introduction:

🚀 Challenge Yourself with Advanced JavaScript Control Structures! #InteractiveQuiz 🧠

Dive into our latest #CodingExercise that focuses on mastering control structures in JavaScript. Build an #InteractiveQuiz application from the ground up, using #IfElse, #Loops, and #TryCatch to create a dynamic and engaging user experience.

👉 Challenge Highlights:

- Dynamic Question Handling
- Real-Time Feedback Based on User Answers
- Comprehensive Use of JavaScript Control Structures

#JavaScript #WebDevelopment #ProgrammingChallenge #CodingSkills #FrontEndDevelopment #LearnToCode #DeveloperCommunity

Join the fun, test your skills, and share your quiz app with our community. Excited to see how you tackle this challenge! 💻🎓🔍 #CodeNewbies #DevLife #TechCommunity

Objective: Develop an interactive quiz application in JavaScript that demonstrates the use of various control structures, including conditional statements (if/else, switch), loops (for, while), and error handling (try/catch). This quiz will present questions to the user and provide feedback based on their answers.

Requirements:

- Use HTML to create the basic structure of the quiz.
- Use CSS for styling.
- Implement JavaScript to handle the quiz logic, scoring, and user interactions.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Interactive Quiz</title>
    <link rel="stylesheet" href="style.css">
</head>
```

```
<body>
    <div id="quizContainer">
        <h2 id="question"></h2>
        <div id="options" class="options-container"></div>
        <button id="nextButton" onclick="nextQuestion()">Next Question</button>
        <div id="result"></div>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
#quizContainer {
    width: 500px;
    margin: auto;
    text-align: center;
    border: 1px solid #ddd;
    padding: 20px;
}
.options-container button {
    display: block;
    margin: 10px auto;
    padding: 5px 15px;
}
```

JavaScript (script.js):

```
const quizQuestions = [
   { question: "What is the capital of France?", options: ["Paris", "London",
"Berlin", "Madrid"], answer: "Paris" },
   { question: "What is 2 + 2?", options: ["3", "4", "5", "6"], answer: "4" },
   // Add more questions here
];
let currentQuestionIndex = 0;
let score = 0;
function displayQuestion() {
   const questionObj = quizQuestions[currentQuestionIndex];
   document.getElementById('question').innerText = questionObj.question;
   const optionsContainer = document.getElementById('options');
   optionsContainer.innerHTML = '';
   questionObj.options.forEach(option => {
      const button = document.createElement('button');
      button.innerText = option;
      button.onclick = () => checkAnswer(option);
      optionsContainer.appendChild(button);
   });
}
function checkAnswer(selectedOption) {
   const correctAnswer = quizQuestions[currentQuestionIndex].answer;
```

```
    if (selectedOption === correctAnswer) {
        score++;
        alert('Correct!');
    } else {
        alert('Wrong answer.');
    }
    document.getElementById('nextButton').disabled = false;
}
function nextQuestion() {
    currentQuestionIndex++;
    if (currentQuestionIndex < quizQuestions.length) {
        displayQuestion();
    } else {
        document.getElementById('quizContainer').innerHTML = `<h1>Quiz
Completed</h1><p>Your score: ${score}</p>`;
    }
    document.getElementById('nextButton').disabled = true;
}
displayQuestion();
```

Explanation:

HTML: Sets up the quiz container with placeholders for the question, options, and result.

CSS: Styles the quiz container and option buttons.

JavaScript:

quizQuestions array stores the questions, options, and answers.

displayQuestion function populates the current question and its options as buttons.

checkAnswer function compares the selected option with the correct answer and updates the score.

nextQuestion function increments the question index and displays the next question, or shows the final score if the quiz is completed.

Control structures like if/else and for loop are used to handle the quiz logic and iterate over the questions and options.

Advanced Coding Exercise: Dynamic Multiplication Table Generator

Introduction:

🌟 Dive Into Advanced JavaScript Loops with Our Latest Coding Challenge! 🔄

Ready to enhance your #JavaScript skills? Take on our #DynamicMultiplicationTableGenerator challenge! This exercise is perfect for mastering different types of loops including #ForLoop, #WhileLoop, and #DoWhileLoop.

🚀 What You'll Do:

- Create an interactive multiplication table generator
- Implement various loop structures for dynamic DOM manipulation
- Sharpen your logic and problem-solving skills

#CodingChallenge #WebDevelopment #JavaScriptLoops #FrontEndDevelopment #ProgrammingFun #LearnToCode #DeveloperCommunity

Join this exciting challenge, share your results, and let's explore the power of loops in JavaScript together! Can't wait to see your amazing tables! 🧮💻🎓 #CodeNewbies #DevLife #TechCommunity

Objective: Develop a JavaScript application that dynamically generates a multiplication table based on user input. This exercise will emphasize the use of various loop structures, including for, while, and do...while loops, to manipulate DOM elements based on user-defined criteria.

Requirements:

Use HTML to create the interface for user input and table display.

Implement JavaScript to generate the multiplication table.

Apply CSS for basic styling.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Multiplication Table Generator</title>
    <link rel="stylesheet" href="style.css">
</head>
```

```
<body>
    <input type="number" id="tableSize" placeholder="Enter table size">
    <button onclick="generateTable()">Generate Table</button>
    <div id="tableContainer"></div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
#tableContainer {
    margin-top: 20px;
}
table {
    width: 100%;
    border-collapse: collapse;
}
table, th, td {
    border: 1px solid black;
    text-align: center;
}
th, td {
    padding: 5px;
}
```

JavaScript (script.js):

```
function generateTable() {
    const tableSize = parseInt(document.getElementById('tableSize').value);
    if (isNaN(tableSize) || tableSize <= 0) {
        alert('Please enter a valid table size.');
        return;
    }
    let tableHTML = '<table><tr>';
    // Generating the header row using a for loop
    for (let i = 0; i <= tableSize; i++) {
        tableHTML += `<th>${i}</th>`;
    }
    tableHTML += '</tr>';
    // Generating table rows using nested for loops
    for (let row = 1; row <= tableSize; row++) {
        tableHTML += `<tr><th>${row}</th>`;
        for (let col = 1; col <= tableSize; col++) {
            tableHTML += `<td>${row * col}</td>`;
        }
        tableHTML += '</tr>';
    }
    tableHTML += '</table>';
    document.getElementById('tableContainer').innerHTML = tableHTML;
```

}

Explanation:

HTML: Provides input for the table size and a button to trigger the table generation, along with a container for displaying the table.

CSS: Styles the multiplication table for better readability.

JavaScript:

generateTable function gets the size of the table from the input, validates it, and generates the table.

The header row of the table is created using a for loop.

Nested for loops are used to generate the rows and columns of the multiplication table.

The generated HTML for the table is then inserted into the tableContainer div.

Advanced Coding Exercise: Custom Array Manipulation Tool

Introduction:

🚀 Unleash the Power of JavaScript Arrays! #AdvancedArrayManipulation 🌟

Step up your coding game with our #JavaScriptArrayManipulationTool challenge. This exercise is designed to deepen your understanding of array methods like #Filter, #Map, #Reduce, and #Sort.

🔥 Features of the Challenge:

- Dynamic array processing

- Implementing and chaining complex array operations

- Enhancing problem-solving and logical thinking

#CodingChallenge #WebDevelopment #JavaScriptArrays #ProgrammingSkills #FrontEndDevelopment #LearnJavaScript #CodeNewbie #DeveloperCommunity

Embrace this challenge and showcase your array manipulation prowess! Excited to see your innovative solutions and creative approaches. Let's code and share! 💻🎯📈 #TechCommunity #DevLife #CodingIsFun

Objective: Create a JavaScript application that allows users to perform various advanced operations on arrays, such as filtering, mapping, reducing, and sorting. This exercise will highlight the use of array methods and how to chain them for complex data manipulation.

Requirements:

Use HTML to create the user interface for array input and operation selection.

Implement JavaScript to handle array operations dynamically.

Apply CSS for basic styling.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <title>Array Manipulation Tool</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <textarea id="arrayInput" placeholder="Enter array elements separated by
commas"></textarea>
    <select id="operation">
        <option value="filter">Filter (keep even numbers)</option>
        <option value="map">Map (square each number)</option>
        <option value="reduce">Reduce (sum all numbers)</option>
        <option value="sort">Sort (ascending order)</option>
    </select>
    <button onclick="performOperation()">Perform Operation</button>
    <div id="result">Result will be shown here</div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}
```

```
textarea, select, button {
    margin: 10px 0;
    display: block;
}
#result {
    margin-top: 20px;
}
```

JavaScript (script.js):

```
function performOperation() {
    const arrayInput = document.getElementById('arrayInput').value;
    const operation = document.getElementById('operation').value;
    const array = arrayInput.split(',').map(item => parseInt(item, 10));
    let result;
    switch (operation) {
        case 'filter':
            result = array.filter(num => num % 2 === 0);
            break;
        case 'map':
            result = array.map(num => num * num);
            break;
        case 'reduce':
            result = array.reduce((acc, curr) => acc + curr, 0);
            break;
```

```
        case 'sort':
            result = array.sort((a, b) => a - b);
            break;
        default:
            result = 'Invalid operation';
    }
    document.getElementById('result').innerText = 'Result: ' + result.join(', ');
}
```

Explanation:

HTML: Provides a textarea for array input, a dropdown for operation selection, and a button to perform the operation.

CSS: Applies basic styling to the elements for better user experience.

JavaScript:

performOperation function retrieves the user's array input and selected operation.

The input string is converted to an array of numbers using split and map.

A switch statement is used to apply the chosen operation (filter, map, reduce, sort) to the array.

The result is displayed in the result div.

Advanced Coding Exercise: Dynamic Travel Destination Matcher

Introduction:

🌍 Explore Advanced JavaScript with the Travel Destination Matcher Challenge! #DynamicArrays 🚀

Ready to take your #JavaScript skills on a journey? Our #TravelDestinationMatcher challenge lets you dive deep into array manipulation. Use #ArrayMethods like filter, map, and every to match travelers with their perfect getaway!

🛤️ Challenge Features:

- Dynamic user preference handling
- Complex array operations for data processing
- Real-world application of JavaScript arrays

#CodingChallenge #WebDevelopment #JavaScriptSkills #ArrayManipulation #SoftwareDevelopment #TechCommunity #CodeNewbies

Join this exciting adventure and share your unique destination matcher! Can't wait to see how you bring this application to life. Let's code, explore, and share!

💻✈️🏖️ #DevelopersParadise #CodingFun #DevLife

Objective: Develop a JavaScript application that matches users with travel destinations based on their preferences. This exercise will emphasize advanced array operations, including filtering, mapping, reducing, and finding elements within arrays.

Requirements:

Use HTML to create a form where users can select their travel preferences.

Implement JavaScript to process these preferences and suggest destinations.

Apply CSS for basic styling.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Travel Destination Matcher</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="preferenceForm">
        <h2>Select Your Travel Preferences</h2>
        <label><input type="checkbox" value="beach" /> Beach</label>
        <label><input type="checkbox" value="mountain" /> Mountain</label>
        <label><input type="checkbox" value="city" /> City</label>
        <label><input type="checkbox" value="culture" /> Culture</label>
        <button onclick="matchDestinations()">Find Destinations</button>
    </div>
    <div id="suggestions">Your travel suggestions will appear here</div>
```

```
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
#preferenceForm, #suggestions {
    width: 300px;
    margin: auto;
    text-align: center;
    border: 1px solid #ddd;
    padding: 20px;
    margin-top: 20px;
}
label {
    display: block;
    margin-top: 10px;
}
```

JavaScript (script.js):

```
const destinations = [
    { name: "Bali", tags: ["beach", "culture"] },
    { name: "New York City", tags: ["city", "culture"] },
    { name: "Alps", tags: ["mountain"] },
```

```
    // ... more destinations
];
function matchDestinations() {
    const selectedTags = Array.from(document.querySelectorAll('#preferenceForm
input:checked')).map(el => el.value);
    const matchingDestinations = destinations.filter(destination =>
        selectedTags.every(tag => destination.tags.includes(tag))
    );
    displaySuggestions(matchingDestinations);
}
function displaySuggestions(destinations) {
    const suggestionsElement = document.getElementById('suggestions');
    if (destinations.length === 0) {
        suggestionsElement.innerHTML = 'No matching destinations found.';
        return;
    }
    const suggestionsHTML = destinations.map(destination =>
`<p>${destination.name}</p>`).join('');
    suggestionsElement.innerHTML = suggestionsHTML;
}
```

Explanation:

HTML: Provides a form with checkboxes for different travel preferences and a div to display suggested destinations.

CSS: Styles the form and suggestions div.

JavaScript:

The destinations array stores objects representing each destination and its associated tags.

matchDestinations function retrieves the selected preferences, filters the destinations array to find matches, and calls displaySuggestions to show the results.

Array methods Array.from, map, filter, and every are used to process the user's selections and match destinations.

Advanced Coding Exercise: Interactive Book Library Manager

Introduction:

📚 Master JavaScript Objects with the Book Library Manager Challenge! #ObjectOrientedProgramming 🚀

Elevate your coding skills by building an interactive #BookLibraryManager in JavaScript. This advanced challenge focuses on object creation, manipulation, and data display. Perfect for those looking to delve deeper into #JavaScriptObjects and #DOMManipulation.

📖 What you'll do:

- Create and manage book objects
- Implement dynamic data addition and display
- Sharpen your JavaScript object handling skills

#CodingChallenge #WebDevelopment #ObjectManipulation #FrontEndDevelopment #ProgrammingSkills #JavaScriptDevelopment #DeveloperCommunity

Objective: Develop a JavaScript application for managing a book library. This exercise emphasizes using objects to store and manipulate data, including object creation, iteration, properties, and methods.

Requirements:

Use HTML to create a user interface for adding and displaying books in the library.
Implement JavaScript to handle book object creation, addition, and display.
Apply CSS for basic styling.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Book Library Manager</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="bookForm">
```

```
        <input type="text" id="title" placeholder="Title">
        <input type="text" id="author" placeholder="Author">
        <button onclick="addBook()">Add Book</button>
    </div>
    <div id="library"></div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
#bookForm, #library {
    width: 300px;
    margin: auto;
    text-align: center;
    border: 1px solid #ddd;
    padding: 20px;
    margin-top: 20px;
}
#library div {
    margin-top: 10px;
    padding: 5px;
    border-bottom: 1px solid #ddd;
}
```

JavaScript (script.js):

```
let library = [];
function Book(title, author) {
    this.title = title;
    this.author = author;
}
function addBook() {
    const title = document.getElementById('title').value;
    const author = document.getElementById('author').value;
    const newBook = new Book(title, author);
    library.push(newBook);
    displayLibrary();
}
function displayLibrary() {
    const libraryDiv = document.getElementById('library');
    libraryDiv.innerHTML = '';
    library.forEach(book => {
        const bookDiv = document.createElement('div');
        bookDiv.innerText = `Title: ${book.title}, Author: ${book.author}`;
        libraryDiv.appendChild(bookDiv);
    });
}
```

Explanation:

HTML: Provides input fields for adding a new book and a div to display the book library.

CSS: Styles the form and library display.

JavaScript:

The Book function constructor creates book objects.

addBook retrieves book details from the form, creates a new book object, adds it to the library array, and updates the display.

displayLibrary iterates over the library array and displays each book in the library div.

This exercise showcases the use of objects, constructors, arrays, and DOM manipulation in JavaScript.

Advanced Coding Exercise: Dynamic To-Do List Application

Introduction:

You'll build a to-do list app that allows adding, removing, and toggling tasks.

📝 Features:

- Interactive task addition

- Task completion toggling

- Deletion of tasks

- Pure #VanillaJavaScript implementation

#CodingChallenge #WebDevelopment #FrontEndDevelopment

#JavaScriptProgramming #UserInteractivity #DeveloperCommunity

Join this challenge, share your innovative to-do list apps, and let's discuss the intricacies of DOM manipulation together. Excited to see your implementations! 💻📝🛠️ #TechCommunity #CodeNewbies #DevLife #BuildInPublic

Objective: Develop a JavaScript application to create a dynamic to-do list. This exercise will emphasize DOM manipulation, including creating, appending, modifying, and deleting DOM elements based on user interactions.

Requirements:

Use HTML to set up the initial structure for the to-do list.

Implement JavaScript to add, remove, and mark tasks as completed.

Apply CSS for basic styling.

HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>To-Do List</title>
    <link rel="stylesheet" href="style.css">
```

```
</head>
<body>
    <div id="todoApp">
        <input type="text" id="newTask" placeholder="Add a new task...">
        <button onclick="addTask()">Add Task</button>
        <ul id="taskList"></ul>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (style.css):

```
#todoApp {
    width: 300px;
    margin: auto;
    text-align: center;
}
#taskList {
    list-style-type: none;
    padding: 0;
}
.task {
    margin-top: 10px;
    padding: 10px;
```

```
    border: 1px solid #ddd;
    background-color: #f9f9f9;
}
.completed {
    text-decoration: line-through;
}
```

JavaScript (script.js):

```
function addTask() {
    const taskInput = document.getElementById('newTask');
    const taskText = taskInput.value.trim();
    if (taskText === '') {
        alert('Please enter a task.');
        return;
    }
    const taskList = document.getElementById('taskList');
    const taskItem = document.createElement('li');
    taskItem.className = 'task';
    taskItem.textContent = taskText;
    taskItem.onclick = function() {
        this.classList.toggle('completed');
    };
    const deleteButton = document.createElement('button');
    deleteButton.textContent = 'Delete';
```

```
    deleteButton.onclick = function() {
        taskList.removeChild(taskItem);
    };
    taskItem.appendChild(deleteButton);
    taskList.appendChild(taskItem);
    taskInput.value = '';
}
// Initial setup
document.getElementById('newTask').addEventListener('keypress', function(e) {
    if (e.key === 'Enter') {
        addTask();
    }
});
```

Explanation:

HTML: Sets up a basic structure with an input field for new tasks, an 'Add Task' button, and an unordered list to display tasks.

CSS: Styles the to-do list app, individual tasks, and completed tasks.

JavaScript:

addTask function: Creates a new task element, adds it to the list, and sets up event handlers for marking as completed and deleting.

Task items toggle the 'completed' class on click to mark or unmark them as completed.

A delete button is added to each task to remove the task from the list.

Adds an event listener to the input field to handle adding tasks on pressing the 'Enter' key.

🎯 Hone Your DOM Manipulation Skills with Our Dynamic To-Do List Challenge! #JavaScriptDOM 🚀

Eager to improve your front-end skills? Dive into our #DynamicToDoList challenge! This exercise is perfect for those looking to master #DOMManipulation in #JavaScript.