### **<https://shorturl.at/oYHye>**

<https://www.telerik.com/blogs/understanding-utilizing-javascript-set-object>

const set = new Set(["value1", "value2"]); // Output: Set(2) { 'value1', 'value2' }

console.log(set.has("value1")); // Output: true

console.log(set.has("value3")); // Output: false

set.delete("value1"); // Output: Set(1) { 'value2' }

set.add("value1"); // Output: Set(2) { 'value1', 'value2' }

set.size; // Output: 2

// How to convert as Array?

const arr1 = Array.from(set); // Output: ['value1', 'value2']

const arr2 = [...mySet]; // Output: ['value1', 'value2']

// How to get value?

set.forEach((value) => { // Method 1

console.log(value);

});

for (const value of set.values()) { // Method 2

console.log(value);

}

// Output for both methods (1 & 2):

// 'value1'

// 'value2'

### 

### **Exercise 1: Flatten Product Categories**

**Task:**

* **Goal:** Turn a nested structure of categories and subcategories into a flat list of category names.
* **Input Example:**

const categories = [

{

name: "Electronics",

subcategories: [

{ name: "Mobile Phones" },

{ name: "Laptops" }

]

},

{

name: "Home Appliances",

subcategories: [

{ name: "Refrigerators" },

{ name: "Washing Machines" }

]

}

];

**Outputs:**

**[**

**'Electronics',**

**'Mobile Phones',**

**'Laptops',**

**'Home Appliances',**

**'Refrigerators',**

**'Washing Machines'**

**]**

**How to Solve:**

* Use a single loop to go through each category.
* For each category, add its name and the names of its subcategories to a new list.

### **Exercise 2: Aggregate Sales Data**

**Task:**

* **Goal:** Calculate the total sales amount for each product based on given records.
* **Input Example:**

const salesData = [

{ productId: 1, sales: [{ amount: 100 }, { amount: 200 }] },

{ productId: 2, sales: [{ amount: 150 }, { amount: 50 }] },

{ productId: 1, sales: [{ amount: 300 }] }

];

**How to Solve:**

* Use an object to keep track of total sales for each product.
* Loop through each sale record and sum the amounts in the sales array for each product.

// Outputs:

// { '1': 600, '2': 200 }

### **Exercise 3: Merge Customer Orders**

**Task:**

* **Goal:** Combine orders from the same customer to find their total spending.
* **Input Example:**

const orders = [

{ customerId: 1, amount: 250 },

{ customerId: 2, amount: 150 },

{ customerId: 1, amount: 200 },

{ customerId: 3, amount: 300 }

];

**How to Solve:**

* Create an object to store total amounts by customer ID.
* Loop through each order and add the amount to the corresponding customer’s total.

### **// Outputs:**

### **// { '1': 450, '2': 150, '3': 300 }**

### **Exercise 4: Retrieve Unique Products**

**Task:**

* **Goal:** Generate a list of unique product IDs from multiple orders.
* **Input Example:**

const orders = [

{ products: [1, 2, 3] },

{ products: [2, 4] },

{ products: [1, 5] }

];

**How to Solve:**

* Use a single loop to iterate through orders.
* Use a Set or an object to keep track of unique product IDs.

### **// Outputs:**

### **// [ 1, 2, 3, 4, 5 ]**

### 

### 

### **Exercise 5: Create a Discount Summary**

**Task:**

* **Goal:** List all products that have a discount.
* **Input Example:**

const products = [

{ id: 1, name: "Laptop", discount: 10 },

{ id: 2, name: "Phone", discount: 0 },

{ id: 3, name: "Tablet", discount: 5 }

];

**How to Solve:**

* Create a new array to hold discounted products.
* Loop through the products and add those with a discount greater than zero to the new array.

// Outputs:

[

{ id: 1, name: 'Laptop', discount: 10 },

{ id: 3, name: 'Tablet', discount: 5 }

]

### 

### **Exercise 6: Deeply Nested Order Summary**

**Task:**

* **Goal:** Create a summary of total order amounts for each customer, consolidating orders that may have multiple items across different nested levels.
* **Input Example:**

const orders = [

{ customerId: 1, orderDetails: [{ items: [{ productId: 101, amount: 50 }, { productId: 102, amount: 150 }] }] },

{ customerId: 2, orderDetails: [{ items: [{ productId: 103, amount: 200 }] }] },

{ customerId: 1, orderDetails: [{ items: [{ productId: 101, amount: 100 }] }] },

{ customerId: 3, orderDetails: [{ items: [{ productId: 104, amount: 300 }, { productId: 105, amount: 150 }] }] }

];

**Output Example:**

{

1: 300,

2: 200,

3: 450

}

**How to Solve:**

* **Initialize a Summary Object**: Create an empty object to hold the total amounts for each customer.
* **Single Loop Iteration**: Use a loop (e.g., forEach) to iterate through each order in the orders array.
* **Extract Amounts**: For each order, iterate through the orderDetails array and further through the nested items array to extract the amount values.
* **Sum the Amounts**: Maintain a running total for each customer using the summary object.
* **Update the Summary Object**: After calculating the total for each order, update the corresponding customer's total in the summary object.

### **Exercise 7: Product Availability Across Multiple Warehouses**

**Task:**

* **Goal:** Determine the total stock available for each product across various warehouses, even when data is deeply nested.
* **Input Example:**

const warehouses = [

{ warehouseId: 1, stock: [{ productId: 101, quantity: 50 }, { productId: 102, quantity: 20 }] },

{ warehouseId: 2, stock: [{ productId: 101, quantity: 30 }, { productId: 103, quantity: 15 }] },

{ warehouseId: 3, stock: [{ productId: 102, quantity: 10 }, { productId: 104, quantity: 5 }] }

];

**Output Example:**

{

101: 80,

102: 30,

103: 15,

104: 5

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object that will store the accumulated stock quantities for each product.
2. **Single Loop Iteration**: Use a loop (such as forEach) to iterate through each warehouse in the warehouses array.
3. **Nested Iteration**: For each warehouse, iterate through its stock array to access each product's productId and quantity.
4. **Accumulate Stock Quantities**:
   * For each product, check if it already exists in the result object.
   * If the product is not yet in the object, initialize it with the current quantity.
   * If it already exists, add the current quantity to the existing total.
5. **Returning the Result**: After processing all warehouses, the result object will contain the total quantities for each product.

### 

### **Exercise 8: Consolidate Customer Feedback**

**Task:**

* **Goal:** Aggregate feedback ratings from multiple customers for various products, where each customer can have multiple feedback entries nested deeply.
* **Input Example:**

const feedbacks = [

{ customerId: 1, feedbackDetails: [{ productId: 101, ratings: [5, 4] }, { productId: 102, ratings: [2] }] },

{ customerId: 2, feedbackDetails: [{ productId: 101, ratings: [3, 4, 5] }] },

{ customerId: 3, feedbackDetails: [{ productId: 102, ratings: [1, 3] }, { productId: 103, ratings: [5] }] }

];

**Output Example:**

{

101: { totalRatings: 17, count: 5, average: 3.4 },

102: { totalRatings: 6, count: 3, average: 2.0 },

103: { totalRatings: 5, count: 1, average: 5.0 }

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object to store the aggregated feedback for each product.  
   **Single Loop Iteration**: Use a loop (e.g., forEach) to iterate through each customer feedback entry in the feedbacks array.  
   **Nested Iteration**: For each customer's feedback, iterate through the feedbackDetails array to access each productId and its associated ratings.
2. **Aggregate Ratings**:
   * For each rating, calculate the total ratings and count for each product.
   * If the product does not exist in the result object yet, initialize its entry.
   * Update the total ratings and count accordingly.
3. **Calculate Average**: After processing all feedback, compute the average rating for each product by dividing the total ratings by the count.
4. **Return the Result**: Finally, the result object will contain the aggregated feedback for each product.

### 

### **Exercise 9: Deep Inventory Analysis**

**Task:**

* **Goal:** Analyze the entire inventory across multiple categories and subcategories to get a count of items in stock.
* **Input Example:**

const inventory = [

{ category: "Electronics", subcategories: [{ name: "Phones", items: [{ id: 1, stock: 10 }, { id: 2, stock: 5 }] }] },

{ category: "Laptops", subcategories: [{ name: "Gaming", items: [{ id: 3, stock: 2 }] }] },

{ category: "Home Appliances", subcategories: [{ name: "Refrigerators", items: [{ id: 4, stock: 0 }, { id: 5, stock: 3 }] }] }

];

**Output Example:**

{

1: 10,

2: 5,

3: 2,

4: 0,

5: 3

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object that will store the item IDs as keys and their corresponding stock counts as values.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each category in the inventory array.
3. **Nested Iteration**:
   * For each category, iterate through its subcategories array.
   * For each subcategory, iterate through its items array.
4. **Accumulate Stock Counts**:
   * For each item, retrieve its id and stock.
   * Add the id as a key in the result object, with the corresponding stock as the value.
5. **Return or Output the Result**: After processing all categories and subcategories, log or return the result object containing the item IDs and their stock counts.

### **Exercise 10: User Interaction Log Analysis**

**Task:**

* **Goal:** Analyze user interaction logs to get the total number of interactions per product, considering nested structures.
* **Input Example:**

const userInteractions = [

{ userId: 1, interactions: [{ productId: 101, type: "view" }, { productId: 102, type: "add\_to\_cart" }] },

{ userId: 2, interactions: [{ productId: 101, type: "purchase" }] },

{ userId: 1, interactions: [{ productId: 103, type: "view" }, { productId: 101, type: "view" }] }

];

**Output Example:**

{

101: 3,

102: 1,

103: 1

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object that will store the product IDs as keys and their total interaction counts as values.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each user interaction entry in the userInteractions array.
3. **Nested Iteration**:
   * For each user interaction, iterate through its interactions array to access each interaction.
4. **Count Interactions**:
   * For each interaction, retrieve the productId.
   * For each productId, check if it already exists in the result object. If it does, increment its count; if it doesn't, initialize it to 1 (since this is the first interaction).
5. **Return or Output the Result**: After processing all user interactions, log or return the result object containing the product IDs and their interaction counts.

### **Exercise 11: Comprehensive Sales Performance Dashboard**

**Task:**

* **Goal:** Create a dashboard summarizing total sales, total quantity sold, and average sale price for each product across multiple categories and sales records.
* **Input Example:**

const salesRecords = [

{ categoryId: 1, sales: [{ productId: 101, quantity: 2, price: 50 }, { productId: 102, quantity: 1, price: 150 }] },

{ categoryId: 1, sales: [{ productId: 101, quantity: 1, price: 50 }] },

{ categoryId: 2, sales: [{ productId: 103, quantity: 3, price: 200 }] },

{ categoryId: 2, sales: [{ productId: 102, quantity: 2, price: 150 }] }

];

**Output Example:**

{

101: { totalSales: 150, totalQuantity: 3, averagePrice: 50 },

102: { totalSales: 300, totalQuantity: 3, averagePrice: 150 },

103: { totalSales: 600, totalQuantity: 3, averagePrice: 200 }

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object to hold the aggregated sales data for each product, where each key is a productId.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each record in the salesRecords array.
3. **Nested Iteration**:
   * For each sales record, iterate through its sales array to access each sale's details.
4. **Aggregate Sales Data**:
   * For each sale, retrieve the productId, quantity, and price.
   * Update the result object:
     + If the product does not exist in the object, initialize an entry with totalSales, totalQuantity, and averagePrice.
     + Update the totalSales by adding the product of quantity and price.
     + Add the quantity to totalQuantity.
     + Store the total number of sales transactions to calculate the average later.
5. **Calculate Average Prices**: After processing all records, compute the average price for each product by dividing totalSales by totalQuantity.
6. **Return or Output the Result**: Finally, log or return the result object containing the summarized sales data for each product.

### **Exercise 12: User Engagement Analysis**

**Task:**

* **Goal:** Analyze user engagement data to determine the number of unique products interacted with and the total interactions for each user.
* **Input Example:**

const userEngagements = [

{ userId: 1, interactions: [{ productId: 101, type: "view" }, { productId: 102, type: "add\_to\_cart" }, { productId: 101, type: "view" }] },

{ userId: 2, interactions: [{ productId: 101, type: "view" }, { productId: 103, type: "purchase" }] },

{ userId: 1, interactions: [{ productId: 103, type: "view" }, { productId: 104, type: "view" }] }

];

**Output Example:**

{

1: { uniqueProducts: 3, totalInteractions: 4 },

2: { uniqueProducts: 2, totalInteractions: 2 }

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object to hold the results, where each key is a userId.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each entry in the userEngagements array.
3. **Track Interactions**:
   * For each user, initialize a Set to track unique productIds and a counter for total interactions.
   * If the user is not already in the result object, add an entry for the user with the initialized Set and interaction count.
4. **Nested Iteration**: For each user's interactions, iterate through the interactions array.
5. **Update Counts**:
   * For each interaction, add the productId to the Set to ensure uniqueness.
   * Increment the total interactions counter for each interaction.
6. **Store Results**: After processing all interactions for a user, calculate the number of unique products (size of the Set) and store both the unique product count and total interactions in the result object for that user.
7. **Return or Output the Result**: Finally, log or return the result object containing the summarized engagement data for each user.

### **Exercise 13: Product Bundling Opportunities**

**Task:**

* **Goal:** Identify potential product bundles based on customer purchases, where each bundle is formed by products frequently bought together.
* **Input Example:**

const purchases = [

{ customerId: 1, items: [{ productId: 101 }, { productId: 102 }] },

{ customerId: 2, items: [{ productId: 101 }, { productId: 103 }] },

{ customerId: 3, items: [{ productId: 102 }, { productId: 103 }] },

{ customerId: 1, items: [{ productId: 104 }, { productId: 101 }] }

];

**Output Example:**

{

'101,102': 2,

'101,103': 2,

'102,103': 1,

'101,104': 1

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object to store the bundle keys (combinations of product IDs) and their respective counts.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each purchase entry in the purchases array.
3. **Nested Iteration**: For each purchase, iterate through the items array to access the productIds.
4. **Generate Combinations**:
   * For each pair of productIds in the purchase, form a combination string (e.g., '101,102').
   * Ensure the combination is sorted to maintain consistency (i.e., always represent pairs in the same order).
5. **Count Occurrences**:
   * Check if the combination string exists in the result object.
   * If it does, increment its count; if not, initialize it to 1.
6. **Return or Output the Result**: After processing all purchases, log or return the result object containing the bundle combinations and their counts.

### **Exercise 14: Customer Loyalty Program Analysis**

**Task:**

* **Goal:** Analyze customer purchase history to categorize customers into loyalty tiers based on total spending.
* **Input Example:**

const customerPurchases = [

{ customerId: 1, orders: [{ amount: 500 }, { amount: 300 }] },

{ customerId: 2, orders: [{ amount: 200 }] },

{ customerId: 3, orders: [{ amount: 700 }, { amount: 100 }] },

{ customerId: 1, orders: [{ amount: 200 }] }

];

**Output Example:**

{

1: 'Gold',

2: 'Silver',

3: 'Gold'

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object to hold the loyalty tier for each customer, where the key is the customerId.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each entry in the customerPurchases array.
3. **Track Total Spending**:
   * For each customer, initialize a variable to track their total spending.
   * If the customer is not already in the result object, add them with their initialized total spending.
4. **Nested Iteration**: For each customer, iterate through their orders array.
5. **Sum Up Amounts**:
   * For each order, add the amount to the customer's total spending.
6. **Determine Loyalty Tier**:
   * After calculating the total spending for each customer, categorize them into tiers based on predefined thresholds:
     + For example:
       - Gold: Spending >= 700
       - Silver: Spending >= 300 and < 700
       - Bronze: Spending < 300
   * Store the corresponding tier in the result object for each customer.
7. **Return or Output the Result**: Finally, log or return the result object containing each customer's loyalty tier.

### **Exercise 15: Complex Product Review Aggregation**

**Task:**

* **Goal:** Aggregate product reviews to calculate average ratings, total reviews, and identify the most frequent review comments.
* **Input Example:**

const reviews = [

{ productId: 101, reviews: [{ rating: 5, comment: "Excellent" }, { rating: 4, comment: "Very Good" }] },

{ productId: 102, reviews: [{ rating: 3, comment: "Average" }, { rating: 3, comment: "Average" }] },

{ productId: 101, reviews: [{ rating: 4, comment: "Very Good" }] },

{ productId: 103, reviews: [{ rating: 5, comment: "Excellent" }] }

];

**Output Example:**

{

101: { averageRating: 4.33, totalReviews: 3, mostFrequentComment: "Very Good" },

102: { averageRating: 3.00, totalReviews: 2, mostFrequentComment: "Average" },

103: { averageRating: 5.00, totalReviews: 1, mostFrequentComment: "Excellent" }

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object to hold the aggregated data for each product, where each key is a productId.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each entry in the reviews array.
3. **Track Ratings and Counts**:
   * For each product, initialize variables to track the total rating, total reviews count, and a frequency map for comments.
   * If the product is not already in the result object, add it with initialized values.
4. **Nested Iteration**: For each product, iterate through its reviews array.
5. **Update Totals and Frequencies**:
   * For each review, add the rating to the total rating.
   * Increment the total reviews count.
   * Track the frequency of each comment using an object to count occurrences.
6. **Calculate Average Rating**: After processing all reviews for a product, calculate the average rating by dividing the total rating by the total reviews count.
7. **Determine Most Frequent Comment**:
   * Iterate through the frequency map to find the comment with the highest count.
8. **Store Results**: Store the average rating, total reviews, and the most frequent comment in the result object for each product.
9. **Return or Output the Result**: Finally, log or return the result object containing the aggregated review data for each product.

### **Exercise 16: Advanced Customer Journey Mapping**

#### **Task Overview**

Create a comprehensive mapping of customer journeys through various touchpoints, summarizing:

* Total interactions
* Number of unique products viewed or purchased
* Average feedback rating for each customer

const customerData = [

{

customerId: 1,

journeys: [

{

interactions: [{ productId: 101, type: "view" }, { productId: 102, type: "add\_to\_cart" }],

purchases: [{ productId: 101, amount: 100 }],

feedback: [{ productId: 101, rating: 5 }]

},

{

interactions: [{ productId: 103, type: "view" }],

purchases: [],

feedback: []

}

]

},

{

customerId: 2,

journeys: [

{

interactions: [{ productId: 101, type: "view" }],

purchases: [{ productId: 102, amount: 150 }],

feedback: [{ productId: 102, rating: 4 }]

},

{

interactions: [{ productId: 104, type: "view" }, { productId: 105, type: "add\_to\_cart" }],

purchases: [{ productId: 105, amount: 200 }],

feedback: [{ productId: 104, rating: 3 }]

}

]

}

];

#### **Expected Output**

{

1: {

totalInteractions: 3,

uniqueProducts: 3,

averageFeedback: {

101: 5.0

}

},

2: {

totalInteractions: 4,

uniqueProducts: 4,

averageFeedback: {

102: 4.0,

104: 3.0

}

}

}

**How to Solve:**

1. **Initialize a Result Object**: Create an empty object to hold the aggregated data for each customer, where each key is a customerId.
2. **Single Loop Iteration**: Use a loop (like forEach) to iterate through each entry in the customerData array.
3. **Track Interactions and Products**:
   * For each customer, initialize variables to track total interactions (count of interaction actions), a Set for unique product IDs, and an object to track feedback ratings.
4. **Nested Iteration**:
   * For each customer, iterate through their journeys array.
   * For each journey, first count the interactions and add the unique product IDs from the interactions array.
   * Then, extract product IDs from the purchases array and add them to the unique product set.
   * Finally, for each feedback entry, update the feedback ratings in the average feedback object.
5. **Calculate Average Feedback**:
   * After processing all journeys for a customer, iterate through the feedback ratings object to calculate the average rating for each product.
6. **Store Results**: Store the total interactions, number of unique products, and the average feedback in the result object for each customer.
7. **Return or Output the Result**: Finally, log or return the result object containing the aggregated customer journey data.