### **<https://shorturl.at/F7CmV>**

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

### **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:**

* Use a single loop to iterate through orders.
* Within that loop, use a method like flatMap to flatten the order details and sum the amounts.

### **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:**

* Use a single loop to go through warehouses.
* Use a single object to accumulate stock 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:**

* Use a single loop to iterate through feedbacks.
* Flatten the ratings using flatMap and aggregate totals and counts.

### 

### **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:**

* + Use a single loop to iterate through categories.
  + Flatten the items and accumulate stock counts in a single object.

### **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: 4,

102: 1,

103: 1

}

**How to Solve:**

* Use a single loop to iterate through user interactions.
* Count interactions for each product in a single object.

### **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:**
  + Use a single loop to iterate through sales records.
  + Maintain an object to aggregate total sales, quantities, and calculate average prices.

### 

### **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: 5 },

2: { uniqueProducts: 2, totalInteractions: 2 }

}

* **How to Solve:**
  + Use a single loop to process each user's engagements.
  + Use a Set to track unique product IDs and count total interactions.

### 

### **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:**

* Use a single loop to combine items into a string key and count occurrences using an object.

### **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'

}

### **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:**

* Use a single loop to aggregate ratings and counts, and a separate structure (e.g., a Map) to track comment frequencies.

### **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

}

}

}