### [**https://shorturl.at/F2bUE**](https://shorturl.at/F2bUE)

### **Exercise 1: Product Fetch with Error Handling**

**Task:** Create a function that fetches product data and handles errors gracefully.

**Function Name:** fetchProductData

**Guidelines:**

* Simulate fetching product data from an API using a Promise.
* If the product ID is invalid (e.g., less than 1), reject the Promise with an error message.
* Log the product details when the Promise resolves successfully.

**Expected Output:**

* Log the product details if fetched successfully.
* Log an error message if the product ID is invalid.

// Answer:

function fetchProductData(productId) {

return new Promise((resolve, reject) => {

if (productId < 1) {

reject(`Invalid product ID: ${productId}`);

return;

}

// Simulate fetching product data

resolve({

id: productId,

name: "Laptop",

price: 999.99,

inStock: true

});

});

}

// Example usage:

fetchProductData(1).then(product => {

console.log('Fetched product:', product);

}).catch(error => {

console.log(error);

});

### **Exercise 2: Bulk Product Fetch with Error Handling**

**Task:** Fetch multiple products using Promise.all with error handling for individual products.

**Function Name:** fetchProducts

**Guidelines:**

* Create a function that accepts an array of product IDs.
* Use Promise.all to fetch all products concurrently, and use above fetchProductData to fetch a product
* If a product fails to fetch (e.g., invalid ID), log the error but ensure the other products are fetched successfully.

**Expected Output:**

* Log a successful fetch for each product.
* Log any errors encountered during the fetch.

### 

### **Exercise 3: Updating Product Inventory with Promise**

**Task:** Create a function to update the inventory of a product.

**Function Name:** updateProductInventory(productId, quantity)

**Guidelines:**

* Create a function that accepts a product ID and the quantity to add or subtract.
* If the product ID is invalid, reject the Promise with an error message.
* If the operation is successful, resolve the Promise with a success message.

// Simulated product data

const products = [

{ id: 1, name: "Laptop", price: 999.99, inStock: 10 },

{ id: 2, name: "Smartphone", price: 699.99, inStock: 5 },

{ id: 3, name: "Wireless Headphones", price: 199.99, inStock: 0}

];

**Expected Output:**

* Log a success message if the inventory is updated.
* Log an error message if the product ID is invalid.

### 

### **Exercise 4: Fetching User Data and Their Posts**

**Objective:** Create a function that fetches user data and their corresponding posts concurrently using Promise.all. Ensure that both fetch operations must complete before proceeding.

**Instructions:**

1. **Simulate Fetching Functions:**
   * fetchUserData(userId)
     + Create a function fetchUserData(userId) that simulates fetching user data from an API. It returns a Promise that resolves with an object representing the user after a delay of **1 second**. The resolved user object should contain the properties: id, which is the user ID passed as an argument, and name, which is a string representing the user's name (e.g., "John Doe").
     + After the delay, log the message: "Fetched user data for user ID: [userId]".
   * fetchUserPosts(userId)
     + Create another function fetchUserPosts(userId) that simulates fetching posts for a specific user. It returns a Promise that resolves with an array of post objects after a delay of **1.5 seconds**. Each post object should include postId, a unique identifier for the post, and content, a string representing the content of the post (e.g., "Hello World!", "Learning JavaScript!").
     + **Success Message:** After the delay, log the message: "Fetched posts for user ID: [userId]".
2. **Implement fetchUserDetails:**
   * Create an async function called fetchUserDetails(userId) that:
     + Calls both fetchUserData(userId) and fetchUserPosts(userId) concurrently using Promise.all.
     + Waits for both Promises to resolve.
     + Logs the user data and their posts.
     + **Success Message:** Log the user data and posts with messages:
       1. "User Data: [userData]"
       2. "User Posts: [userPosts]".
     + **Error Message:**
       1. Log the message: "Error fetching user details: [error message]" by using a try/catch block
3. **Example Usage:**
   * Call fetchUserDetails(1) to demonstrate the functionality with a valid user ID. You can also test error handling by calling fetchUserDetails(-1) or by modifying one of the fetch functions to reject the Promise under certain conditions.

### **Exercise 5: Fetch Product Data**

### **Objective: Write a function to fetch product data using async/await.**

### **Instructions:**

### **Create an async function called getProduct.**

### **Use await to call fetchProductData from Exercise 1.**

### **Log the fetched product details.**

### 

### 

### 

### **Exercise 6: Add to Cart**

**Objective**: Create a function that allows users to add products to the shopping cart.

**Instructions**:

1. Using below simulated product data.

### **const products = [**

### **{ id: 1, name: "Laptop", price: 999.99, inStock: 10 },**

### **{ id: 2, name: "Smartphone", price: 699.99, inStock: 5 },**

### **{ id: 3, name: "Wireless Headphones", price: 199.99, inStock: 0 },**

### **];**

1. Define a global variable cart as an empty array to hold the items in the shopping cart.
2. Create an async function called addToCart that takes productId and quantity as parameters.
3. Inside the function:
   * Use the find method to check if the product exists in the products array.
   * If the product does not exist, log an error message.
   * If the product exists, check if there is enough stock to fulfill the requested quantity.
   * If there isn't enough stock, log an error message.
   * If valid, check if the product is already in the cart. If it is, update the quantity; if not, add it to the cart.
4. Log a success message indicating how many of the product were added to the cart.

// Example usage

await addToCart(1, 2); // Add 2 laptops to the cart

### 

### **Exercise 7: View Cart**

**Objective**: Create a function to display the contents of the shopping cart.

**Instructions**:

1. Using below simulated product data.

### **const products = [**

### **{ id: 1, name: "Laptop", price: 999.99, inStock: 10 },**

### **{ id: 2, name: "Smartphone", price: 699.99, inStock: 5 },**

### **{ id: 3, name: "Wireless Headphones", price: 199.99, inStock: 0 },**

### **];**

1. Define an async function called viewCart.
2. Inside the function, initialize a total variable to accumulate the total price of the items in the cart.
3. Loop through the cart array using forEach to display each item's name, quantity, and total price for that item.
4. After the loop, calculate and display the overall total price of the cart.

### 

### 

### **Exercise 8: Mock Payment Function**

### **Objective:** Create a function to simulate payment processing.

**Instructions**:

1. **Define the Function**:
   * Create an async function named processPayment that takes totalAmount as a parameter.
2. **Simulate Delay**:
   * Use setTimeout to simulate a 1-second delay for payment processing.
3. **Return a Promise**:
   * Inside the setTimeout, return a promise that resolves with a success message:
     + "Payment of $<totalAmount> processed successfully."
   * Ensure totalAmount is formatted to two decimal places.
4. **Usage**:
   * Call this function in your application to simulate processing a payment.

### 

### **Exercise 9: Sequential Fetch with Controlled Flow**

**Task:** Fetch products sequentially and log the details after each fetch.

**Function Name:** fetchProductsSequentially

**Guidelines:**

* Create an async function that takes an array of product IDs.
* Use a [delay function](https://medium.com/@brandon.lau86/how-to-create-a-delay-in-javascript-and-a-practical-use-of-async-await-4dbd1d7744f4) to simulate a wait time (e.g., 1 second) between each fetch.
* Log the details of each product as they are fetched.

**Expected Output:**

* Log each product after a delay between fetches.