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

### **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.

### **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

**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.

**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: 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.