Project: Product Inventory Manager

This hands-on will guide you through building a Product Inventory Manager using advanced JavaScript concepts. You will work with dates, regular expressions, callbacks, setInterval, Promises, Fetch API, and higher-order array functions. Each step builds on the previous one, culminating in a functional product inventory manager.

Part 1: JS Dates

- Create a function **calculateDateDifference** that takes two date strings in the format **YYYY-MM-DD HH:MM:SS** and returns the difference between them in days, hours, and minutes.
- Use the Date object to parse the dates and calculate the difference.
- Log the result to the console in the format: **X days, Y hours, Z minutes.**

Example:

```
console.log(calculateDateDifference("2023-10-05 14:30:45",
"2023-10-10 10:15:30"));
// Example output: "4 days, 19 hours, 44 minutes"
```

Part 2: JS Regular Expressions

- Create a function **validateProductSKU** that checks if a product SKU is valid.
- A valid SKU should:
    - Start with 3 uppercase letters (representing the product category).
    - Followed by a hyphen.
    - End with 6 digits (representing the product ID).
- Use a regular expression to validate the SKU.
- Log the result to the console.

Example:

```
console.log(validateProductSKU("ABC-123456")); // true
console.log(validateProductSKU("123-ABCDEF")); // false
```

Part 3: JS Callbacks

- Create a function **processOrder** that takes three arguments:
    - orderId (string): The ID of the order.
    - callback (function): A callback function to execute after processing the order.
    - delay (number): The delay in milliseconds before executing the callback.
- Inside the function, simulate processing the order by logging the order ID and the current date and time.
- Use setTimeout to execute the callback after the specified delay.
- The callback should simulate additional steps like payment confirmation and shipping notification.

Example:

```
processOrder("ORDER123", (order) => {
  console.log(`Payment confirmed for order: ${order}`);
  setTimeout(() => {
    console.log(`Order shipped: ${order}`);
  }, 2000);
}, 3000);
```

Part 4: JS Intervals

- Create a function **monitorInventory** that takes three arguments:
    - productId (string): The ID of the product.
    - callback (function): A callback function to execute repeatedly.
    - interval (number): The interval in milliseconds between executions.
- Use setInterval to execute the callback repeatedly at the specified interval.
- Log the product ID and the current date and time each time the callback runs.
- Simulate random stock level changes (e.g., increase or decrease by 1-5 units) and pass the updated stock level to the callback.

Example:

```
monitorInventory("PROD123", (stock) => console.log(stock), 5000);
// Output every 5 seconds: { productId: "PROD123", stock: 15 }
```

Part 5: JS Promises

- Create a function **restockProduct** that returns a promise.
- The promise resolves if the restocking is successful.
- The promise rejects if the restocking fails (e.g., due to supplier issues).
- Simulate an asynchronous operation using setTimeout.
- Log the result of the promise using both .then/.catch and async/await.

Example:

```
restockProduct("PROD123")
  .then(() => console.log("Product restocked successfully!"))
  .catch(() => console.log("Failed to restock product..."));

// Using async/await
(async () => {
  try {
    await restockProduct("PROD123");
    console.log("Product restocked successfully!");
  } catch {
    console.log("Failed to restock product...");
  }
})();
```

Part 6: Fetch API

- Use the Fetch API to fetch product data from a public API (e.g., FakeStoreAPI).
- Fetch a list of products and filter them based on:
  - Price range (e.g., between 10 and 50).
  - Category (e.g., "electronics").
- Log the filtered product names and prices.

Part 7: Higher-Order Array Methods

- Use the fetched products from Step 6.
- Calculate and log the following statistics:
    - Total number of products.
    - Average price of products.
    - Most expensive product.
    - Cheapest product.
- Use higher-order array functions like **reduce**, **map**, and **sort**.