1. **Variables, Data Types, and String Manipulation**

Create a function that accepts user input (name, age, and email), validates the input types, and returns a formatted string: "Name: [name], Age: [age], Email: [email]". Ensure the email includes "@".

1. **Arrays & Object:**

Create a JavaScript object representing a library system. The object should include properties for the library name and an array of books. Each book object should have the following properties: title, author, isbn, and isAvailable (a Boolean indicating availability).

Implement the following functions that work with this library object:

* **Add a Book**:  
  Create a function addBook(library, book) that adds a new book to the books array of the library. Console log "Book added" after adding the book.
* **Check Out a Book**:  
  Create a function checkOutBook(library, isbn) that sets isAvailable to false for the book with the specified isbn. If the book is not found, display "Book not found in the library".
* **Return a Book**:  
  Create a function returnBook(library, isbn) that sets isAvailable to true for the book with the specified isbn. If the book is not found, display "Book not found in the library".
* **List All Book Titles**:  
  Create a function listAllBookTitles(library) that lists all book titles using the map method. Use forEach to print each title to the console.
* **List Available Books**:  
  Create a function listAvailableBooks(library) that lists titles of only the available books using filter and map. Print each available book title to the console.
* **Count Books by Author**:  
  Create a function countBooksByAuthor(library, author) that uses reduce to count how many books are authored by the given author. Return the count.

**Hints:**

* + Use the **map** method to extract book titles and the **forEach** method to print them.
  + Use the **filter** method to find available books before using map.
  + Use the **reduce** method to accumulate the count of books by a specific author

1. **ES6+ Features**

Write a function that takes an object with nested properties (e.g., user profile) and uses destructuring to extract specific properties. Use the spread operator to create a new object with some updated properties.

1. **Asynchronous JavaScript & Error Handling**

Write a JavaScript function that performs the following tasks:

* **Fetch Data**:  
  Use async/await to fetch data from the public API endpoint **https://jsonplaceholder.typicode.com/posts**.
* **Process and Validate Data**:  
  Extract the title property from each post and create an array of titles. Validate this array to ensure it contains only strings.
* **Error Handling**:  
  Handle any errors that occur during the fetch operation or data processing. Log an appropriate error message if something goes wrong.

**Function Requirements:**

* **Function Signature**: async function fetchAndProcessTitles()
* **Use async/await** for fetching data and handle errors with try/catch.
* **Validate** that the titles array contains only strings.
* **Log** the array of titles or an error message as appropriate.

**Advanced Questions**

1. **Callbacks and Promises**

Simulate a series of asynchronous operations (e.g., fetching user data, fetching user's posts, and fetching user's comments). Use callbacks to handle each operation, and then refactor the code to use Promises and async/await.

1. **Real-World Scenario 1: User Data Processing**

Write a function that processes an array of user data. Each user object has properties id, name, age, and isActive. The function should:

Filter out inactive users.

Sort the remaining users by age in descending order.

Return an array of user names.

1. **Real-World Scenario 2: Order Processing**
   1. Create a function that calculates the total order amount from an array of order objects. Each order has properties orderId, customerId, and amount. The function should:
      1. Sum the amounts.
      2. Return the total order amount.
      3. Handle empty arrays and invalid data gracefully.
2. **Real-World Scenario 3: Simulating API Calls**
   1. Write a function that simulates fetching user data from an API. Use setTimeout to mimic the delay. Return user data after 1 second. Ensure error handling is in place, and use async/await for the final implementation.