# Common JS Module Pattern : module.exports and require, Destructuring Object and Arrays & Callback Functions

## 1. Basic Module Export and Require

**Problem Statement:**

**Create a module called `math.js` that exports a function `add` which takes two numbers and returns their sum. In a separate file (`main.js`), require the `math.js` module and use the `add` function to calculate the sum of `5` and `10`. Log the result.**

**Expected Output:**

```
Sum: 15
```

## 2. Export Multiple Functions

**Problem Statement:**

**Create a module called `stringOperations.js` that exports the following functions:**

- **`toUpperCase(str)`: Converts the string to uppercase.**
- **`toLowerCase(str)`: Converts the string to lowercase.**

**In a separate file (`main.js`), require the `stringOperations.js` module and use the functions to convert the string `"Hello World"` to uppercase and lowercase. Log the results.**

**Expected Output:**

```
Uppercase: HELLO WORLD
Lowercase: hello world
```

## 3. Export an Object

**Problem Statement:**

**Create a module called `config.js` that exports an object with the following properties:**

- `appName`: "MyApp"
- `version`: "1.0.0"
- `author`: "John Doe"

In a separate file (`main.js`), require the `config.js` module and log the `appName`, `version`, and `author`.

**Expected Output:**

```
App Name: MyApp
Version: 1.0.0
Author: John Doe
```

### 4. Destructure Object Properties

**Problem Statement:**
**Given the following object:**

```
const person = {
  name: "Alice",
  age: 25,
  occupation: "Engineer"
}
```

**Use object destructuring to extract the `name`, `age`, and `occupation` properties. Log the extracted values.**

**Expected Output:**

```
Name: Alice
Age: 25
Occupation: Engineer
```

### 5. Destructure Nested Object Properties

**Problem Statement:**
**Given the following nested object:**

```
const user = {
  id: 1,
  fullName: {
```

```
    firstName: "John",
    lastName: "Doe"
  },
  contact: {
    email: "john.doe@example.com",
    phone: "123-456-7890"
  }
}
```

**Use object destructuring to extract `firstName`, `lastName`, `email`, and `phone`. Log the extracted values.**

**Expected Output:**

```
First Name: John
Last Name: Doe
Email: john.doe@example.com
Phone: 123-456-7890
```

### 6. Destructure Array Elements

**Problem Statement:**
**Given the following array:**

```
const fruits = ["Apple", "Banana", "Cherry"];
```

**Use array destructuring to extract the first, second, and third elements. Log the extracted values.**

**Expected Output:**

```
First Fruit: Apple
Second Fruit: Banana
Third Fruit: Cherry
```

### 7. Destructure Array with Skipping Elements

**Problem Statement:**
**Given the following array:**

```
const numbers = [10, 20, 30, 40, 50];
```

**Use array destructuring to extract the first, third, and fifth elements. Log the extracted values.**

**Expected Output:**

```
First Number: 10
Third Number: 30
Fifth Number: 50
```

### 8. Simple Callback

**Problem Statement:**

Create a function called `greet` that takes two arguments:

- `name`: A string (e.g., "John").
- `callback`: A callback function that takes the `name` as an argument and logs a greeting message.

Call the `greet` function with the name "John" and a callback that logs:

```
Hello, John!
```

**Expected Output:**

```
Hello, John!
```

### 9 Callback with Array Processing

**Problem Statement:**

Create a function called `processArray` that takes two arguments:

- `array`: An array of numbers (e.g., `[1, 2, 3, 4, 5]`).
- `callback`: A callback function that processes each element of the array (e.g., multiplies it by 2).

Call the `processArray` function with the array `[1, 2, 3, 4, 5]` and a callback that multiplies each element by 2. Log the processed array.

**Expected Output:**

```
[2, 4, 6, 8, 10]
```

### 10. Callback with Filtering

**Problem Statement:**

Create a function called `filterArray` that takes two arguments:

- `array`: An array of numbers (e.g., `[10, 20, 30, 40, 50]`).

- `callback`: A callback function that filters the array based on a condition (e.g., returns numbers greater than 25).

Call the `filterArray` function with the array `[10, 20, 30, 40, 50]` and a callback that filters numbers greater than 25. Log the filtered array.

**Expected Output:**

```
[30, 40, 50]
```