Week 1 Lab Activities: Introduction to DRY, SOLID, Coupling, and Cohesion

---

Objective:

By the end of this lab, students will be able to:

- Understand the **DRY** principle and how to avoid code repetition.
- Implement the **SOLID** principles in both JavaScript and Python.
- Analyse and reduce **coupling** between components.
- Increase **cohesion** within modules and functions.

Lab Activities Overview

1. **Activity 1: DRY Principle**
    - **Language**: JavaScript
    - **Time**: 30 minutes
    - **Goal**: Demonstrate how to refactor code to remove repetition.
2. **Activity 2: Implementing SOLID Principles**
    - **Language**: Python
    - **Time**: 45 minutes
    - **Goal**: Implement key SOLID principles (Single Responsibility Principle and Open/Closed Principle) in Python code.
3. **Activity 3: Understanding Coupling and Cohesion**
    - **Language**: JavaScript
    - **Time**: 30 minutes
    - **Goal**: Refactor code to reduce coupling and increase cohesion.
4. **Conclusion**: A brief 15-minute Q&A and reflection session on what students learned.

Activity 1: DRY Principle in JavaScript

Overview:

Students will start by identifying repeated code in a JavaScript snippet and refactor it to make the code adhere to the **DRY** principle.

Steps:

1. Provide students with the following JavaScript code that has repetitions:

   ```
   js
   Copy code
   function getAreaOfSquare(side) {
       return side * side;
   }

   function getAreaOfCircle(radius) {
       return 3.1416 * radius * radius;
   }

   function getAreaOfTriangle(base, height) {
       return 0.5 * base * height;
   }

   function getAreaOfRectangle(length, width) {
       return length * width;
   }
   ```

2. **Task**: Refactor this code to remove repetition by using a single function that takes a shape and its dimensions as input and returns the area.
3. **Discussion**: After completing the refactoring, students should explain why the new version is more maintainable and adheres to the DRY principle.

Activity 2: Implementing SOLID Principles in Python

Overview:

This activity will focus on implementing the **Single Responsibility Principle (SRP)** and **Open/Closed Principle (OCP)** in Python.

Steps:

1. Poorly designed Python code:

   ```
   python
   Copy code
   class Report:
       def __init__(self, data):
           self.data = data

       def print_report(self):
           print("Report:")
           for item in self.data:
               print(f"Item: {item}")

       def save_to_file(self, filename):
           with open(filename, 'w') as file:
               for item in self.data:
                   file.write(f"Item: {item}\n")
   ```

2. **Task (SRP)**: Refactor the class `Report` to follow the **Single Responsibility Principle**, by separating the reporting logic from the saving logic.

3. **Task (OCP)**: Modify the `Report` class to make it **open for extension, but closed for modification**. Introduce a new feature that allows reports to be saved in multiple formats (e.g., JSON, plain text) without modifying the original `save_to_file` method directly.

4. **Discussion**: After refactoring, students should explain how this design adheres to the Open/Closed Principle and is more flexible for future extensions.

Activity 3: Understanding Coupling and Cohesion in JavaScript

Overview:

Students will review a code sample that exhibits high coupling and low cohesion and will refactor it to reduce coupling and improve cohesion.

Steps:

1. Tghtly coupled JavaScript code:

   ```
   js
   Copy code
   function displayUserProfile(user) {
       console.log(`Name: ${user.name}`);
       console.log(`Email: ${user.email}`);
       if (user.subscription === 'premium') {
           console.log('Premium user');
       } else {
           console.log('Standard user');
       }
   }

   function sendUserNotification(user) {
       if (user.subscription === 'premium') {
           console.log('Sending premium notification to', user.email);
       } else {
           console.log('Sending standard notification to', user.email);
       }
   }
   ```

2. **Task**: Refactor this code to reduce **coupling** between the `displayUserProfile` and `sendUserNotification` functions. There are two things you need to do to complete this.
3. **Discussion**: Analyse why the new design improves modularity, reduces coupling, and increases cohesion between related functionalities.

Conclusion (15 minutes):

- **Reflection & Q&A**: Ask students to reflect on:
    - How their refactored code adheres to **DRY**, **SOLID**, **coupling**, and **cohesion** principles.
    - Real-world scenarios where these principles would be particularly important.