Practical 2: Callback Functions and Callback Hell

Aim

To write a JavaScript program that demonstrates the use of callback functions for handling asynchronous operations. The program executes multiple dependent tasks - login verification, cart verification, payment processing, and order placement - in a specific sequence, demonstrating callback hell and the limitations of nested callbacks.

Theory

A callback function is a function passed as an argument to another function, which is then invoked after the completion of an asynchronous operation.

When multiple async operations depend on each other, callbacks must be nested inside one another. This creates deeply indented, hard-to-read code known as "Callback Hell" or the "Pyramid of Doom".

Program: E-Commerce Order Flow using Callbacks

```
function checkLogin(callback) {
    setTimeout(() => {
        console.log("Login successful");
        callback();
    }, 2000);
}

function verifyCart(callback) {
    setTimeout(() => {
        console.log("Cart verified");
        callback();
    }, 2000);
}

function proceedToPayment(callback) {
    setTimeout(() => {
        console.log("Payment successful");
        callback();
    }, 2000);
}

function placeOrder(callback) {
    setTimeout(() => {
        console.log("Order placed");
        callback();
    }, 2000);
}
```

Callback Hell - Nested Execution

Each step depends on the previous one completing first. This forces us to nest callbacks inside each other, creating the pyramid structure:

Practical 2: Callback Functions and Callback Hell

```
checkLogin(() => {
    verifyCart(() => {
        proceedToPayment(() => {
            placeOrder(() => {
                console.log("All steps completed!");
            });
        });
    });
});
```

Execution Flow

1. checkLogin() runs -> waits 2 seconds -> prints "Login successful"
2. verifyCart() runs -> waits 2 seconds -> prints "Cart verified"
3. proceedToPayment() runs -> waits 2 seconds -> prints "Payment successful"
4. placeOrder() runs -> waits 2 seconds -> prints "Order placed"
5. Final callback prints "All steps completed!"

Total execution time: ~8 seconds (each step waits for the previous one)

Output:

```
Login successful          (after 2s)
Cart verified             (after 4s)
Payment successful        (after 6s)
Order placed              (after 8s)
All steps completed!
```

Conclusion

Callback functions allow us to handle asynchronous operations in sequence. However, when multiple dependent async tasks are chained, the code becomes deeply nested and hard to read/maintain - this is called Callback Hell. Modern JavaScript solves this problem using Promises and async/await syntax.