Debugging JavaScript Variable Scope

Function: testVariables()

This JavaScript function demonstrates how var, let, and const behave within different scopes.

Code:

```
function testVariables() {
    var x = 10;
    let y = 20;
    const z = 30;

    if (true) {
        var x = 40;      // Reassigns the function-scoped x (10 -> 40)
        let y = 50;      // New block-scoped 'y'
        // const z = 60; // Would be allowed as new block-scoped 'z'

        console.log(x); // Logs: 40
        console.log(y); // Logs: 50
        console.log(z); // Logs: 30
    }

    console.log(x); // Logs: 40 (overwritten inside 'if')
    console.log(y); // Logs: 20 (original let y)
    console.log(z); // Logs: 30 (original const z)
}
```

```
testVariables();
```

Output:

```
-------------------------------------------------
40
50
30
40
20
30
```

Key Concepts:

---

1. var is function-scoped, so re-declaring it inside a block affects the whole function.
2. let and const are block-scoped, so re-declaring them inside a block does not affect the outer variable.
3. The const z is accessible inside the 'if' block because it is not re-declared inside.

Conclusion:

Understanding JavaScript's variable scope rules is crucial for avoiding bugs related to variable shadowing and scope leakage.