**Solidity Functions Modifiers**

**Introduction:**

In this lesson, we will explore modifiers and how they can simplify code writing and management in Solidity. Modifiers enable developers to create reusable code snippets that can be applied to multiple functions, enhancing code readability, maintainability, and security.

**Repeated Conditions:**

If we build a contract with multiple administrative functions, that should only be executed by the contract owner, we might repeatedly check the caller identity:

Require(msg.sender == owner, “Sender is not owner”);

However, repeating this line in every function clutters the contract, making it harder to read, maintain, and debug.

**Modifiers:**

Modifiers in Solidity allow embedding custom lines of code within any function to modify its behaviour.

Here’s how to create a modifier:

Modifier onlyOwner {

Require(msg.sender == owner, “Sender is not owner”);

\_;

}

**Note:** The modifier is named `onlyOwner` to reflect the condition it checks.

**The `\_` (underscore):**

The underscore `\_` placed in the body is a placeholder for the modified function’s code. When the function with the modifier is called, the code before `\_` runs first, and if it succeeds, the function’s code executes next.

For example, the `onlyOwner` modifier can be applied to the `withdraw` function like this:

Function withdraw(uint amount) public onlyOwner {

// Function logic }

When `withdraw` is called, the contract first executes the `onlyOwner` modifier. If the `require` statement passes, the rest of the `withdraw` function executes.

If the underscore `\_` were placed before the `require` statement, the function’s logic would execute first, followed by the `require` check, which is not the intended use case.

**Conclusion:**

Using modifiers like `onlyOwner` simplifies contract development by centralizing common conditions, reducing code repetition, and enhancing contract readability and maintainability.