**Using Safemath**

**Introduction:**

In this lesson, we will explore `SafeMath`, a widely used library before Solidity version 0.8, and understand why its usage has now decreased.

**Integer Overflow:**

`SafeMath.sol` was a staple in Solidity contracts before version 0.8. After this version, its usage has significantly dropped.

Let’s begin by creating a new file called `SafeMathTester.sol` and adding a function `add` that increments the `bigNumber` state variable.

// SafeMathTester.sol

Pragma solidity ^0.6.0;

Contract SafeMathTester {

Uint8 public bigNumber = 255;

Function add() public {

bigNumber = bigNumber + 1;

}

}

Notice we are using compiler version `0.6.0`. The `bigNumber` is a `uint8` variable with a maximum value of `255`. If we call the `add` function, it will return `0` instead of the expected `256`.

Notice we are using compiler version `0.6.0`. The `bigNumber` is a `uint8` variable with a maximum value of `255`. If we call the `add` function, it will return `0` instead of the expected `256`.

Before Solidity version 0.8.0, signed and unsigned integers were unchecked, meaning that if they exceeded the maximum value the variable type could hold, they would reset to the lower limit. This pattern is known as integer overflow and the `SafeMath` library was designed to prevent it.

**SafeMath:**

`SafeMath.sol` provided a mechanism to revert transactions when the maximum limit of a `uint256` data type was reached. It was a typical security measure across contracts to avoid erroneous calculations and potential exploits.

Function add(uint a, uint b) public pure returns (uint) {

Uint c = a + b;

Require(c >= a, “SafeMath: addition overflow”);

Return c;

}

**Solidity 0.8.0:**

With the introduction of Solidity version 0.8, automatic checks for overflows and underflows were implemented, making `SafeMath` redundant for these checks. If `SafeMathTester.sol` is deployed with Solidity `0.8.0`, invoking the `add` function will cause a transaction to fail, when, in older versions, it would have reset to zero.

For scenarios where mathematical operations are known not to exceed a variable’s limit, Solidity introduced the `unchecked` construct to make code more gas-efficient. Wrapping the addition operation with `unchecked` will ignore the overflow and underflow checks: if the `bigNumber` exceeds the limit, it will wrap its value to zero.

Uint8 public bigNumber = 255;

Function add() public {

Unchecked {

bigNumber = bigNumber + 1;

}

}

**Caution:** It’s important to use unchecked blocks with caution as they reintroduce the possibility of overflows and underflows.

**Conclusion:**

The evolution of Solidity and `SafeMath.sol` highlights the continuous advancements in Ethereum smart contract development. Although recent updates have made `SafeMath.sol` less essential, it remains a significant part of Ethereum’s history. Understanding its role provides valuable insight into the progress and maturation of Solidity.