Error

An error will undo all changes made to the state during a transaction.

You can throw an error by calling `require`, `revert` or `assert`.

- `require` is used to validate inputs and conditions before execution.
- `revert` is similar to `require`. See the code below for details.
- `assert` is used to check for code that should never be false. Failing assertion probably means that there is a bug.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Error {
    function testRequire(uint _i) public pure {
        // Require should be used to validate conditions such as:
        // - inputs
        // - conditions before execution
        // - return values from calls to other functions
        require(_i > 10, "Input must be greater than 10");
    }

    function testRevert(uint _i) public pure {
        // Revert is useful when the condition to check is complex.
        // This code does the exact same thing as the example above
        if (_i <= 10) {
            revert("Input must be greater than 10");
        }
    }

    uint public num;

    function testAssert() public view {
        // Assert should only be used to test for internal errors,
        // and to check invariants.

        // Here we assert that num is always equal to 0
        // since it is impossible to update the value of num
        assert(num == 0);
    }
}
```

Here is another example

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

contract Account {
    uint public balance;
    uint public constant MAX_UINT = 2**256 - 1;

    function deposit(uint _amount) public {
        uint oldBalance = balance;
        uint newBalance = balance + _amount;

        // balance + _amount does not overflow if balance + _amount >= balance
        require(newBalance >= oldBalance, "Overflow");

        balance = newBalance;

        assert(balance >= oldBalance);
    }

    function withdraw(uint _amount) public {
        uint oldBalance = balance;

        // balance - _amount does not underflow if balance >= _amount
        require(balance >= _amount, "Underflow");

        if (balance < _amount) {
            revert("Underflow");
        }

        balance -= _amount;

        assert(balance <= oldBalance);
    }
}
```

Try on Remix

Take a course at Smart Contract Engineer

Smart Contract Programmer
Donate
0x95a6...fa243
your ETH will directly go to Khan Academy
contact@solidity-by-example.org