Solidity Require Statements Quiz

Quiz 1: Understanding require() Statements

Instructions: You're helping Neri protect a Community Fund from hackers like Hackana. Which of these explains what the `require()` statement does in Solidity?

```
pragma solidity ^0.8.0;

contract CommunityFund {
    address public owner = msg.sender;

    function withdraw(uint256 amount) public {
        require(msg.sender == owner, "Only the owner can withdraw");
        // Code to withdraw funds
    }
}
```

What does the `require()` statement do in this function?

- A) It shows a warning message but lets the function keep running
- B) It checks a condition and cancels the transaction if the condition is false
- C) It's just a comment that reminds developers about a rule
- D) It writes an error message to the blockchain but doesn't stop the function

Answer: B) It checks a condition and cancels the transaction if the condition is false

Explanation: Think of `require()` as a security guard at the entrance of a function. It checks if a condition is true (in this case, if the person calling the function is the owner). If the condition is true, the function continues normally. But if the condition is false, the function immediately stops, all changes are undone, and the error message "Only the owner can withdraw" is returned to the user. This is how Solidity protects functions from attacks like those from Hackana.

Quiz 2: Validating Transaction Values with require()

Instructions: Neri is building a donation system where the amount sent must match what the user specifies. Which function correctly validates this?

```
pragma solidity ^0.8.0;

contract SecureDonation {
    uint256 public totalDonations;

    // Option A
    function donate1(uint256 amount) public payable {
        totalDonations += amount;
    }
```

```
    // Option B
    function donate2(uint256 amount) public payable {
        require(msg.value == amount, "Sent value must match specified amount");
        totalDonations += amount;
    }

    // Option C
    function donate3(uint256 amount) public payable {
        if (msg.value == amount) {
            totalDonations += amount;
        }
    }

    // Option D
    function donate4(uint256 amount) public payable {
        require(amount > 0);
        totalDonations += msg.value;
    }
}
```

Which function correctly ensures the sent value matches the specified amount?

- A) Option A
- B) Option B
- C) Option C
- D) Option D

Answer: B) Option B

Explanation: Option B correctly uses `require()` to check that the amount of Ether sent with the transaction (`msg.value`) exactly matches the `amount` parameter. If someone tries to trick the system by saying they're donating 100 but only sending 50, the transaction will be rejected. Options A and D don't check if the values match at all. Option C uses an if statement instead of require, which means if the values don't match, it silently does nothing without giving any error message or reverting the transaction - this could be confusing for users.

Quiz 3: Multiple Security Checks with require()

Instructions: Neri needs to secure a withdrawal function against multiple types of attacks. Which implementation is the most secure?

```
pragma solidity ^0.8.0;

contract DefenseSystem {
    address public owner;
    uint256 public totalFunds;
    bool public emergencyLockdown = false;

    constructor() {
```

```
        owner = msg.sender;
    }

    // Option A
    function withdraw1(uint256 amount) public {
        require(msg.sender == owner);
        require(amount <= totalFunds);
        totalFunds -= amount;
        payable(msg.sender).transfer(amount);
    }

    // Option B
    function withdraw2(uint256 amount) public {
        require(
            msg.sender == owner &&
            amount <= totalFunds &&
            !emergencyLockdown,
            "Withdrawal not allowed"
        );
        totalFunds -= amount;
        payable(msg.sender).transfer(amount);
    }

    // Option C
    function withdraw3(uint256 amount) public {
        require(msg.sender == owner, "Only owner can withdraw");
        require(amount <= totalFunds, "Insufficient funds");
        require(!emergencyLockdown, "System is locked down");
        totalFunds -= amount;
        payable(msg.sender).transfer(amount);
    }

    // Option D
    function withdraw4(uint256 amount) public {
        if (msg.sender == owner && amount <= totalFunds && !emergencyLockdown) {
            totalFunds -= amount;
            payable(msg.sender).transfer(amount);
        }
    }
}
```

Which option provides the best security with helpful error messages?

- A) Option A
- B) Option B
- C) Option C
- D) Option D

Answer: C) Option C

Explanation: Option C is the most secure with the clearest feedback because it:

1. Uses separate `require()` statements for each condition
2. Includes specific error messages for each possible failure
3. Checks all necessary conditions: owner-only access, sufficient funds, and no emergency lockdown

Having separate require statements with clear messages helps users understand exactly what went wrong if their transaction fails. Option A doesn't check the emergency lockdown and has no error messages. Option B combines all checks into one require statement, making it harder to tell which specific condition failed. Option D uses an if statement that would silently fail without any explanation to the user.

Quiz 4: Preventing Zero-Value Transactions

Instructions: Hackana might try to cause confusion by making zero-value donations. Which function correctly prevents this?

```
pragma solidity ^0.8.0;

contract DonationProtection {
    uint256 public totalDonations;

    // Option A
    function donate1() public payable {
        totalDonations += msg.value;
    }

    // Option B
    function donate2() public payable {
        if (msg.value > 0) {
            totalDonations += msg.value;
        }
    }

    // Option C
    function donate3() public payable {
        require(msg.value != 0, "Donation amount cannot be zero");
        totalDonations += msg.value;
    }

    // Option D
    function donate4() public payable {
        require(msg.value > 0, "Donation amount must be greater than zero");
        totalDonations += msg.value;
    }
}
```

Which function best prevents zero-value donations with a clear message?

- A) Option A
- B) Option B
- C) Option C
- D) Option D

Answer: D) Option D

Explanation: Option D is best because it uses `require(msg.value > 0, "Donation amount must be greater than zero")` which:

1. Ensures the donation is positive (greater than zero)
2. Reverts the transaction if the condition fails
3. Provides a clear error message explaining why the transaction was rejected

Option A accepts zero-value donations without any checks. Option B silently ignores zero-value donations but doesn't inform the user why nothing happened. Option C works similarly to D but using `msg.value != 0` instead of `msg.value > 0` - both prevent zero values, but "must be greater than zero" is slightly clearer language for users to understand.