Solidity View and Pure Functions Quiz

Quiz 1: Understanding View Functions

Instructions: Neri is building a contract for the Barangay San Juan to track service fees. Which function below is correctly using the `view` modifier?

```
pragma solidity ^0.8.0;

contract BarangayServiceFees {
    uint256 public idFee = 100;  // Fee for Barangay ID
    uint256 public permitFee = 50;  // Fee for Business Permit

    // Option A
    function getTotalFee(uint256 idCount, uint256 permitCount) view public returns
(uint256) {
        return (idFee * idCount) + (permitFee * permitCount);
    }

    // Option B
    function setIdFee(uint256 newFee) view public {
        idFee = newFee;
    }

    // Option C
    function getIdFee() view public returns (uint256) {
        idFee += 1;  // Small increase
        return idFee;
    }

    // Option D
    function getPermitFee() public view returns (uint256) {
        return permitFee;
    }
}
```

Which function correctly uses the `view` modifier?

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

Answer: D) Option D

Explanation: Option D correctly uses the `view` modifier because it only reads a state variable (`permitFee`) without modifying any state.

View functions can read state variables but cannot modify them. Think of them like "read-only" functions - they let you look at data but not change anything.

The other options have problems:

- Option A is trying to modify state variables while marked as `view`
- Option B tries to change the `idFee` variable, which isn't allowed in a view function
- Option C tries to increase `idFee` with `+=`, which modifies state and isn't allowed in a view function

Remember: A function should be marked as `view` when it only needs to read (but not modify) the contract's state.

Quiz 2: Understanding Pure Functions

Instructions: Neri is creating functions for a barangay fee calculator. Which function below is correctly using the `pure` modifier?

```
pragma solidity ^0.8.0;

contract FeeCalculator {
    uint256 public baseFee = 50;

    // Option A
    function calculateTotal(uint256 quantity) public pure returns (uint256) {
        return baseFee * quantity;
    }

    // Option B
    function addFees(uint256 fee1, uint256 fee2) public pure returns (uint256) {
        return fee1 + fee2;
    }

    // Option C
    function getBaseFee() public pure returns (uint256) {
        return baseFee;
    }

    // Option D
    function calculateDiscount(uint256 amount) public pure returns (uint256) {
        uint256 discount = 10;
        return amount - (amount * discount / 100);
    }
}
```

Which function correctly uses the `pure` modifier?

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

Answer: B) Option B

Explanation: Option B correctly uses the `pure` modifier because it only works with its input parameters (`fee1` and `fee2`) without reading or modifying any state variables.

Pure functions are even more restricted than view functions:

- They cannot read state variables
- They cannot modify state variables
- They can only work with their input parameters and local variables

Think of pure functions as "math calculators" that only operate on the values you give them.

The other options have issues:

- Option A tries to read `baseFee` (a state variable), which isn't allowed in a pure function
- Option C tries to return `baseFee`, which reads a state variable
- Option D defines a local variable `discount` as a fixed value (10). While this works, it would be more efficient to make discount a parameter instead of hardcoding it

Quiz 3: When to Use View vs. Pure

Instructions: Neri needs to calculate fees for different barangay services. Which function should use a `view` modifier and which should use a `pure` modifier?

```
pragma solidity ^0.8.0;

contract ServiceCalculator {
    // Fee schedule stored in the contract
    uint256 public idFee = 100;
    uint256 public certificationFee = 50;
    uint256 public permitFee = 200;

    // Function 1: Calculate ID fee with senior discount
    function calculateSeniorIdFee() public returns (uint256) {
        return idFee * 80 / 100;  // 20% discount
    }

    // Function 2: Calculate total for multiple services
    function calculateServiceTotal(uint256 idCount, uint256 certCount, uint256
permitCount) public returns (uint256) {
        return (idFee * idCount) + (certificationFee * certCount) + (permitFee *
permitCount);
    }

    // Function 3: Calculate percent of a given amount
    function calculatePercent(uint256 amount, uint256 percent) public returns
(uint256) {
        return amount * percent / 100;
    }
}
```

Which function should use a `pure` modifier?

- A) Function 1 (calculateSeniorIdFee)
- B) Function 2 (calculateServiceTotal)
- C) Function 3 (calculatePercent)
- D) None of these functions should use pure

Answer: C) Function 3 (calculatePercent)

Explanation: Function 3 (`calculatePercent`) should use a `pure` modifier because it only performs a calculation based on its input parameters (`amount` and `percent`) without reading or modifying any contract state variables.

Functions 1 and 2 both need to read state variables (`idFee`, `certificationFee`, `permitFee`), so they should use the `view` modifier instead of `pure`.

Using the appropriate modifiers helps:

1. Make your code's intentions clearer
2. Save gas when these functions are called from outside the blockchain
3. Allow the compiler to catch mistakes if you accidentally try to modify state in functions that shouldn't

Quiz 4: Gas Savings with View and Pure

Instructions: Neri is optimizing her barangay service contract. Which statement is true about gas usage with `view` and `pure` functions?

```
pragma solidity ^0.8.0;

contract GasUsageExample {
    uint256 public serviceCount = 0;

    // Regular function
    function incrementServiceCount() public {
        serviceCount += 1;
    }

    // View function
    function getServiceCount() public view returns (uint256) {
        return serviceCount;
    }

    // Pure function
    function calculateServiceFee(uint256 baseRate, uint256 hours) public pure
returns (uint256) {
        return baseRate * hours;
    }
}
```

Which statement about gas usage is correct?

- A) View and pure functions always cost gas to execute
- B) View and pure functions cost gas only when called from within another function that modifies state
- C) View and pure functions don't cost gas when called externally (off-chain)
- D) Pure functions cost gas but view functions don't

Answer: C) View and pure functions don't cost gas when called externally (off-chain)

Explanation: View and pure functions are special because they don't cost any gas when they're called externally (off-chain) - like from a website or application outside the blockchain. This is because they don't change any data on the blockchain, so they can just return a result without requiring a transaction.

This makes them perfect for:

- Reading information from the blockchain
- Doing calculations based on blockchain data
- Checking conditions before submitting actual transactions

However, there's an important exception: if a view or pure function is called from another function that does modify state (within the same transaction), then it will cost gas as part of that transaction.

For example:

- Calling `getServiceCount()` directly from your application costs no gas
- But if the `incrementServiceCount()` function internally called `getServiceCount()`, then you'd pay gas for both as part of the transaction