Solidity Constructors Quiz

Quiz 1: Understanding Constructors

Instructions: Neri is creating a registration system for a barangay program. Look at the code below and identify what a constructor does in Solidity.

```
pragma solidity ^0.8.0;

contract BarangayProgram {
    string public programName;
    address public programAdmin;

    constructor(string memory _name) {
        programName = _name;
        programAdmin = msg.sender;
    }

    function updateProgramName(string memory _newName) public {
        require(msg.sender == programAdmin, "Only admin can update");
        programName = _newName;
    }
}
```

What is the purpose of the constructor in this contract?

- A) It runs every time someone calls the contract
- B) It runs only once when the contract is first deployed
- C) It runs whenever the updateProgramName function is called
- D) It runs once a day to update the contract

Answer: B) It runs only once when the contract is first deployed

Explanation: A constructor is like the setup instructions that run just once when a contract is created (deployed). In this example, the constructor sets the initial program name and automatically makes whoever deploys the contract the program admin. After the contract is deployed, the constructor never runs again - even if you call other functions in the contract. It's perfect for setting up important initial values that you want to establish right at the beginning.

Quiz 2: Constructor Parameters

Instructions: Neri wants to deploy multiple versions of a donation tracker for different barangays. Which code correctly uses constructor parameters?

```
pragma solidity ^0.8.0;

contract DonationTracker {
```

```
    string public barangayName;
    uint256 public fundingGoal;
    uint256 public totalDonations;

    // Option A
    constructor() {
        barangayName = "San Juan";
        fundingGoal = 10000;
    }

    // Option B
    constructor(string memory _barangayName, uint256 _fundingGoal) {
        barangayName = _barangayName;
        fundingGoal = _fundingGoal;
    }

    // Option C
    function constructor(string memory _barangayName, uint256 _fundingGoal) public
{
        barangayName = _barangayName;
        fundingGoal = _fundingGoal;
    }

    // Option D
    constructor {
        barangayName = input._barangayName;
        fundingGoal = input._fundingGoal;
    }
}
```

Which constructor is correctly written to accept parameters?

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

Answer: B) Option B

Explanation: Option B correctly shows how to create a constructor that accepts parameters. When you deploy this contract, you would need to provide a barangay name and funding goal, and these values would be used to initialize the contract's state variables.

Option A is a valid constructor but doesn't accept any parameters - it always sets the same hardcoded values. Option C incorrectly tries to make a constructor as a regular function, which won't work (constructors are special and don't use the `function` keyword). Option D has incorrect syntax - parameters need to be declared properly within parentheses.

Quiz 3: Constructor Access Control

Instructions: Neri is building a contract that needs to remember who deployed it. What is the most common use of msg.sender in a constructor?

```
pragma solidity ^0.8.0;

contract CommunityProject {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

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

What is msg.sender referring to in the constructor?

- A) The CommunityProject contract itself
- B) The address that deployed the contract
- C) The address that will call the withdrawFunds function
- D) Anyone who interacts with the contract

Answer: B) The address that deployed the contract

Explanation: In a constructor, `msg.sender` refers to the address that deployed (created) the contract. This is commonly used to set up the initial owner or admin of a contract. By storing this address in the `owner` variable, the contract will remember who deployed it, allowing that person special privileges (like withdrawing funds) that others don't have. This is one of the most common patterns in smart contracts for establishing ownership and access control.

Quiz 4: Initializing Variables

Instructions: Neri wants to make sure her contract starts with the correct initial values. Which variables MUST be initialized in the constructor rather than being set as defaults?

```
pragma solidity ^0.8.0;

contract BarangayRegistry {
    string public barangayName;
    uint256 public residentCount = 0;
    address public barangayOfficial;
    bool public isActive = true;

    constructor(string memory _name, address _official) {
        barangayName = _name;
        barangayOfficial = _official;
```

```
    }
}
```

Which variables MUST be initialized in the constructor rather than directly in their declarations?

- A) Both barangayName and barangayOfficial
- B) Only barangayName
- C) Only barangayOfficial
- D) Neither, both could be initialized directly

Answer: A) Both barangayName and barangayOfficial

Explanation: Variables that depend on information that's only available at deployment time (like who's deploying the contract or custom settings) should be initialized in the constructor. In this case:

- `barangayName` needs to be set in the constructor because it's different for each barangay and must be provided when deploying
- `barangayOfficial` needs to be set in the constructor because each barangay has a different official

The other variables (`residentCount` and `isActive`) have fixed initial values that don't depend on deployment conditions, so they can be initialized directly in their declarations.