Solidity Functions Quizzes

Quiz 1: Function Visibility

Instructions: Let's imagine you're building a jeepney payment system. You want to have a function that updates the total money collected, but you want to make sure only the contract itself can use this function. Which visibility option should you use?

```
pragma solidity ^0.8.0;

contract JeepneyPaymentSystem {
    uint256 public baseFare = 12;         // Everyone can see the base fare
    uint256 private totalCollected;       // This is hidden from outside

    // This function updates our money counter
    function updateTotalCollected(uint256 amount) _____ {
        totalCollected += amount;
    }
}
```

Choose the correct option to fill in the blank:

- A) `external` (Can be called from outside but not from inside the contract)
- B) `public` (Can be called from anywhere)
- C) `internal` (Can be called from this contract and contracts that inherit from it)
- D) `private` (Can only be called from inside this contract)

Answer: D) `private`

Explanation: Think of `private` as a personal diary that only you can read. When we use the `private` keyword, only the contract itself can use that function. This is perfect for sensitive operations like updating money amounts. Using `private` here keeps the function safe from being called by other contracts or users, which helps prevent mistakes or misuse.

Quiz 2: Payable Functions

Instructions: You're creating a function that lets jeepney passengers pay their fare using cryptocurrency. The function needs to accept money from passengers. Which option allows your function to receive payments?

```
pragma solidity ^0.8.0;

contract JeepneyFares {
    mapping(address => uint256) public passengerBalances;  // Tracks how much each
passenger has paid

    function payFare() _____ {
        require(msg.value > 0, "Payment amount must be greater than 0");
```

```
        passengerBalances[msg.sender] += msg.value;
    }
}
```

Choose the correct option to fill in the blank:

- A) `public view returns (uint256)` (Function that can only look at data)
- B) `public payable` (Function that can receive money)
- C) `external pure` (Function that doesn't access any contract data)
- D) `private` (Function that can only be called from inside the contract)

Answer: B) `public payable`

Explanation: Think of `payable` as putting a "payment accepted" sign on your function. Without this special keyword, your function can't receive any money (Ether). When passengers want to pay their fare, they need to send cryptocurrency with their transaction, and only `payable` functions can accept this money. The word `public` means anyone can call this function, which is what we want for a payment system where many different passengers need to pay.

Quiz 3: View and Pure Functions

Instructions: In this jeepney fare calculator, one function has a mistake in how it's labeled. Can you spot which function is using the wrong modifier?

```
pragma solidity ^0.8.0;

contract FareCalculator {
    uint256 public distanceBasedFare = 2;  // Price per kilometer stored in the
contract

    // Function A
    function calculateFare(uint256 distance) public pure returns (uint256) {
        return distance * distanceBasedFare;  // Tries to use the stored price
    }

    // Function B
    function getBaseFare() public view returns (uint256) {
        return distanceBasedFare;  // Just looks up the stored price
    }

    // Function C
    function calculateDiscount(uint256 fare, uint256 discountPercent) public pure
returns (uint256) {
        return fare - ((fare * discountPercent) / 100);  // Uses only the inputs
    }

    // Function D
    function getFareWithTax(uint256 distance) public view returns (uint256) {
        uint256 baseFare = distance * distanceBasedFare;
        return baseFare + (baseFare / 10);  // Adds 10% tax
```

```
    }
}
```

Which function has a mistake?

- A) Function A
- B) Function B
- C) Function C
- D) Function D

Answer: A) Function A

Explanation: Function A has a problem! It's marked as `pure`, which means "I don't read or change any data from the contract." But then it tries to use `distanceBasedFare`, which is data stored in the contract!

Think of it like this:

- `pure` functions are like calculators: they only use the numbers you give them
- `view` functions are like reading a price list: they can look at stored values but can't change them

Function A should be marked as `view` instead of `pure` because it needs to read the stored price. Functions B and D are correctly marked as `view` because they read contract data, and Function C is correctly `pure` because it only uses the inputs provided.

Quiz 4: Function Parameters and Return Values

Instructions: You're creating a function that calculates discounted jeepney fares for students. Which of these functions is written correctly to calculate the student discount?

```
pragma solidity ^0.8.0;

contract StudentDiscounts {
    uint256 public studentDiscountPercent = 20;  // Students get 20% off

    // Option A
    function calculateStudentFare(uint baseFare) public view returns (uint) {
        uint discount = (baseFare * studentDiscountPercent) / 100;
        return baseFare - discount;
    }

    // Option B
    function calculateStudentFare(uint baseFare) public view returns {
        uint discount = (baseFare * studentDiscountPercent) / 100;
        return baseFare - discount;
    }

    // Option C
    function calculateStudentFare(uint baseFare) public returns (uint) {
        uint discount = (baseFare * studentDiscountPercent) / 100;
        return baseFare - discount;
    }
```

```
    // Option D
    function calculateStudentFare(uint baseFare) pure public returns (uint) {
        uint discount = (baseFare * studentDiscountPercent) / 100;
        return baseFare - discount;
    }
}
```

Which option is correct?

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

Answer: A) Option A

Explanation: Option A is correct for three simple reasons:

1. It tells us what type of information is coming back with `returns (uint)` - this is like promising to return a number
2. It uses `view` which is right because the function only looks at the discount percentage but doesn't change it
3. The calculation works correctly: it figures out the discount amount and subtracts it from the fare

The other options have mistakes:

- Option B is missing the return type - it doesn't say what kind of value is coming back
- Option C doesn't use `view` even though it's just reading data
- Option D uses `pure` but then tries to read the discount percentage from the contract, which isn't allowed with `pure`

Quiz 5: Function Modifiers Order

Instructions: When writing Solidity functions, the order of keywords matters. Only one of these functions has the keywords in the right order. Which one is it?

```
pragma solidity ^0.8.0;

contract JeepneyOwnerOperations {
    address public owner;  // The jeepney owner's address

    constructor() {
        owner = msg.sender;  // Set the owner to whoever deploys the contract
    }

    // Option A
    function updateFare(uint256 newFare) payable public returns (bool) {
        require(msg.sender == owner, "Only owner can update fare");
        // Code to update fare
```

```
        return true;
    }

    // Option B
    function updateFare(uint256 newFare) public payable returns (bool) {
        require(msg.sender == owner, "Only owner can update fare");
        // Code to update fare
        return true;
    }

    // Option C
    function updateFare(uint256 newFare) returns public payable (bool) {
        require(msg.sender == owner, "Only owner can update fare");
        // Code to update fare
        return true;
    }

    // Option D
    function updateFare(uint256 newFare) public returns payable (bool) {
        require(msg.sender == owner, "Only owner can update fare");
        // Code to update fare
        return true;
    }
}
```

Which function has the keywords in the correct order?

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

Answer: B) Option B

Explanation: Think of function keywords like putting on clothes in the right order:

1. First, who can see it? (`public`, `private`, etc.)
2. Then, can it receive money? (`payable`)
3. Finally, what does it give back? (`returns (bool)`)

Option B has this right order: `public payable returns (bool)`. This is like saying "anyone can call this function, it can receive money, and it gives back a yes/no answer."

The other options mix up this order, which will cause errors when you try to compile your code.

Quiz 6: Function Overloading

Instructions: In Solidity, "function overloading" means having multiple functions with the same name but different inputs. Which example below shows the correct way to do function overloading?

```
pragma solidity ^0.8.0;

contract JeepneyRoutes {
    // Option A
    function calculateFare(uint256 distance) public pure returns (uint256) {
        return distance * 2;  // Basic fare: 2 wei per kilometer
    }

    function calculateFare(uint256 distance, uint256 passengerCount) public pure
returns (uint256) {
        return distance * 2 * passengerCount;  // Group fare: price per person ×
number of people
    }

    // Option B
    function getFareInfo() public pure returns (uint256) {
        return 12;  // Returns the fare as a number
    }

    function getFareInfo() public pure returns (string memory) {
        return "Base fare is 12 wei";  // Returns the fare as text
    }

    // Option C
    function payFare(uint256 amount) public payable {
        // Accepts payment with cryptocurrency
    }

    function payFare(uint256 amount) public {
        // Same inputs but doesn't accept cryptocurrency
    }

    // Option D
    function getTotalFare(uint256 distance) public pure returns (uint256) {
        return distance * 2;
    }

    function calculateTotalFare(uint256 distance) public pure returns (uint256) {
        return distance * 2;
    }
}
```

Which option shows correct function overloading?

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

Answer: A) Option A

Explanation: Option A is correct because it shows proper function overloading.

Think of it like this: we have two functions with the same name `calculateFare`, but they take different inputs:

- One takes just the distance
- The other takes both distance AND passenger count

This is like having two buttons labeled "Calculate Fare" on a calculator, but one asks for one piece of information while the other asks for two pieces.

The other options don't work because:

- Option B tries to change only what the function returns, not what it takes in
- Option C tries to use the same name and same inputs but just change whether it's payable
- Option D uses two completely different function names, which isn't overloading at all