Solidity Arrays Quiz

Quiz 1: Understanding Different Array Types

Instructions: Neri is creating a system to track daily transactions in the market. Which of these correctly declares an array that can hold exactly 7 daily totals?

```
pragma solidity ^0.8.0;

contract MarketRecords {
    // Option A
    uint256[] public weeklyTotals;

    // Option B
    uint256[7] public weeklyTotals;

    // Option C
    array(uint256, 7) public weeklyTotals;

    // Option D
    uint256 public weeklyTotals[7];
}
```

Which option correctly declares a fixed-size array for 7 daily totals?

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

Answer: B) Option B

Explanation: Option B correctly declares a fixed-size array that can hold exactly 7 values. The syntax `uint256[7]` creates an array of unsigned integers with 7 slots.

Think of it like a row of 7 boxes, where each box can hold a number. This is perfect for storing daily totals for a week (7 days).

Option A declares a dynamic array that can grow or shrink, not fixed at 7. Options C and D use incorrect syntax that doesn't exist in Solidity.

Quiz 2: Adding Elements to Arrays

Instructions: Neri wants to record a list of payments received at her market stall. Which code correctly adds a new payment to the list?

```
pragma solidity ^0.8.0;

contract PaymentTracker {
    uint256[] public payments;

    // Option A
    function addPayment1(uint256 amount) public {
        payments.push(amount);
    }

    // Option B
    function addPayment2(uint256 amount) public {
        payments.add(amount);
    }

    // Option C
    function addPayment3(uint256 amount) public {
        payments[payments.length] = amount;
    }

    // Option D
    function addPayment4(uint256 amount) public {
        payments.append(amount);
    }
}
```

Which function correctly adds a payment to the array?

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

Answer: A) Option A

Explanation: Option A correctly adds a new payment to the array using the `.push()` method. This is the standard way to add elements to a dynamic array in Solidity.

Options B and D use methods (`.add()` and `.append()`) that don't exist for arrays in Solidity. Option C attempts to directly assign a value to the index at the array's length, but this won't work because array indices start at 0 and go up to `length-1`, so `payments[payments.length]` would be out of bounds.

Quiz 3: Working with Array Indices

Instructions: Neri needs to update the price of an item in her product list. What's wrong with the following function?

```
pragma solidity ^0.8.0;

contract PriceUpdater {
```

```
    uint256[] public prices = [10, 20, 30, 40, 50];

    function updatePrice(uint256 index, uint256 newPrice) public {
        prices[index] = newPrice;
    }
}
```

What's the problem with this function?

- A) You can't update array values directly, you need to use a special method
- B) The function needs to use push() instead of direct assignment
- C) There's no check to make sure the index is within the array's bounds
- D) Dynamic arrays can't be initialized with values as shown

Answer: C) There's no check to make sure the index is within the array's bounds

Explanation: The problem is that the function doesn't check if the `index` is valid. If someone calls `updatePrice(10, 100)` but the array only has 5 elements (indices 0-4), the transaction will fail with an error.

A better version would include a check like:

```
require(index < prices.length, "Index out of bounds");
```

This would make sure the index is valid before trying to update the price, and provide a helpful error message if it's not.

The other options are incorrect: Option A is wrong because you can update array values directly as shown. Option B is incorrect because `push()` adds new elements, not updates existing ones. Option D is incorrect because dynamic arrays can indeed be initialized with values as shown.

Quiz 4: Understanding Array Length and Storage

Instructions: Neri wants to track the last 10 customers in her store. She wrote the function below, but it has a problem. What is it?

```
pragma solidity ^0.8.0;

contract CustomerTracker {
    address[10] public recentCustomers;
    uint256 public customerCount = 0;

    function addCustomer(address customer) public {
        // Add the new customer
        recentCustomers[customerCount] = customer;

        // Update the counter
        customerCount += 1;
```

```
    }
}
```

What problem might occur with this code?

- A) The array will grow too large and waste storage
- B) The function will fail after 10 customers
- C) New customers will overwrite the oldest ones
- D) Customer addresses could be duplicated in the array

Answer: B) The function will fail after 10 customers

Explanation: The problem is that `customerCount` keeps increasing without limit, but the array has a fixed size of 10. When the 11th customer tries to register, `customerCount` will be 10, and the code will try to access `recentCustomers[10]` - but arrays in Solidity are zero-indexed, so the valid indices are only 0 through 9.

This will cause the transaction to fail with an "index out of bounds" error. To fix this, Neri needs to wrap around to the beginning of the array when she reaches the end, which can be done using the modulo operator:

```
recentCustomers[customerCount % 10] = customer;
```

This would make it cycle through indices 0-9 repeatedly, ensuring that new customers overwrite the oldest ones in a circular buffer pattern.