Solidity Payable Functions Quiz

Quiz 1: Understanding Payable Functions

Instructions: Neri is creating a contract for secure donations to help fight Hackana. Which function correctly allows the contract to receive Ether?

```
pragma solidity ^0.8.0;

contract DonationSystem {
    // Option A
    function donate1() public {
        // Process donation
    }

    // Option B
    function donate2() public payable {
        // Process donation
    }

    // Option C
    function donate3() payable public {
        // Process donation
    }

    // Option D
    function donate4() public view {
        // Process donation
    }
}
```

Which function correctly allows the contract to receive Ether?

- A) donate1()
- B) donate2()
- C) donate3()
- D) donate4()

Answer: B) donate2() and C) donate3()

Explanation: Both Options B and C are correct as they use the `payable` keyword, which allows a function to receive Ether. The order of `public` and `payable` doesn't matter, so both syntax options work.

The other options don't allow Ether to be sent:

- Option A has no `payable` keyword, so it will reject any Ether sent with the transaction
- Option D has the `view` modifier, which indicates a read-only function that doesn't modify state and cannot receive Ether

The `payable` modifier is essential for any function that needs to receive cryptocurrency. Without it, the transaction will be rejected if Ether is sent.

Quiz 2: Accessing Transaction Value

Instructions: Neri needs to track donation amounts in her anti-Hackana campaign. Which code correctly accesses the amount of Ether sent to a payable function?

```
pragma solidity ^0.8.0;

contract DonationTracker {
    mapping(address => uint256) public donations;

    // Option A
    function recordDonation1() public payable {
        donations[msg.sender] += msg.amount;
    }

    // Option B
    function recordDonation2() public payable {
        donations[msg.sender] += msg.value;
    }

    // Option C
    function recordDonation3() public payable {
        donations[msg.sender] += this.balance;
    }

    // Option D
    function recordDonation4() public payable {
        donations[msg.sender] += tx.value;
    }
}
```

Which function correctly tracks the amount of Ether sent by the donor?

- A) recordDonation1()
- B) recordDonation2()
- C) recordDonation3()
- D) recordDonation4()

Answer: B) recordDonation2()

Explanation: Option B correctly uses `msg.value` to access the amount of Ether sent with the current transaction.

The other options are incorrect:

- Option A uses `msg.amount`, which doesn't exist in Solidity

- Option C uses `this.balance`, which gives the total balance of the contract, not the amount sent in the current transaction
- Option D uses `tx.value`, which doesn't exist (the correct global variable is `msg.value`)

In Solidity, `msg.value` is the special global variable that gives you access to the amount of Wei (the smallest unit of Ether) sent along with the function call. This is crucial for tracking individual transaction amounts.

Quiz 3: Fallback and Receive Functions

Instructions: Neri is creating a contract that can accept Ether through direct transfers. Which code snippet correctly implements a receive function?

```
pragma solidity ^0.8.0;

// Option A
contract EtherReceiver1 {
    uint256 public totalReceived;

    function receive() public payable {
        totalReceived += msg.value;
    }
}

// Option B
contract EtherReceiver2 {
    uint256 public totalReceived;

    receive() external payable {
        totalReceived += msg.value;
    }
}

// Option C
contract EtherReceiver3 {
    uint256 public totalReceived;

    receive payable() external {
        totalReceived += msg.value;
    }
}

// Option D
contract EtherReceiver4 {
    uint256 public totalReceived;

    fallback() external payable {
        totalReceived += msg.value;
    }
}
```

Which contract correctly implements a receive function that accepts Ether?

- A) EtherReceiver1
- B) EtherReceiver2
- C) EtherReceiver3
- D) EtherReceiver4

Answer: B) EtherReceiver2

Explanation: Option B correctly implements the `receive()` function according to Solidity's syntax requirements:

- It uses the keyword `receive` (not as a regular function name)
- It has `external` visibility
- It has the `payable` modifier
- It has no arguments and no return values

The other options have issues:

- Option A declares `receive` as a regular function with a function keyword, which is incorrect syntax
- Option C has incorrect syntax with `payable` before the parentheses
- Option D implements a `fallback()` function, which is different from the `receive()` function

The `receive()` function is a special function in Solidity that is automatically called when the contract receives Ether without any data (calldata). It must be marked as `external` and `payable`.

Quiz 4: Using Payable for Contract Interactions

Instructions: Neri is building a payment distribution system for the barangay. Which function correctly sends Ether to another address?

```
pragma solidity ^0.8.0;

contract PaymentDistributor {
    address payable public beneficiary;

    constructor(address payable _beneficiary) {
        beneficiary = _beneficiary;
    }

    function receiveFunds() public payable {
        // Receive funds
    }

    // Option A
    function distribute1() public {
        beneficiary.send(address(this).balance);
    }

    // Option B
    function distribute2() public {
        beneficiary.transfer(address(this).balance);
    }
    // Option C
    function distribute3() public {
        beneficiary = address(this).balance;
    }

    // Option D
    function distribute4() public {
        beneficiary.call{value: address(this).balance}("");
    }
}
```

Which function(s) can correctly send Ether from the contract to the beneficiary?

- A) distribute1() only
- B) distribute2() only
- C) Both distribute1() and distribute2()
- D) distribute1(), distribute2(), and distribute4()

Answer: D) distribute1(), distribute2(), and distribute4()

Explanation: There are three ways to send Ether in Solidity, all of which are represented in the options:

1. `send()` (Option A):
   - Returns a boolean indicating success/failure
   - Has a gas limit of 2300 gas (enough for simple operations only)
   - Does not revert on failure
2. `transfer()` (Option B):
   - Automatically reverts the transaction if the transfer fails
   - Has a gas limit of 2300 gas
   - Recommended for simple transfers where you want to ensure safety
3. `call()` with value (Option D):
   - Most flexible method
   - No built-in gas limit
   - Returns success boolean and data
   - Currently recommended approach for sending Ether

Option C is incorrect as it attempts to assign a uint256 value (the contract's balance) to an address variable, which will cause a type error.

Modern Solidity practice (since 0.8.0) generally recommends using the low-level `call` method with value specified (Option D), but all three valid methods can work depending on the circumstances.