Solidity Ether and Wei Quiz

Quiz 1: Understanding Ether and Wei Conversions

Instructions: Neri needs to counter Hackana's transaction manipulation by understanding Ether units. Which statement about Ether and Wei is correct?

```
pragma solidity ^0.8.0;

contract EtherUnits {
    // Option A
    uint256 public oneEtherInWei = 10**9;

    // Option B
    uint256 public oneEtherInWei2 = 1000000000000000000;

    // Option C
    uint256 public oneEtherInWei3 = 1 ether;

    // Option D
    uint256 public oneEtherInWei4 = 10**12;
}
```

Which option correctly represents 1 Ether in Wei?

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

Answer: C) Option C

Explanation: Option C is correct because `1 ether` in Solidity equals 10^18 Wei, which is the smallest unit in Ethereum.

The other options are incorrect:

- Option A (10^9) actually equals 1 Gwei, not 1 Ether
- Option B (1000000000000000000) is numerically correct but doesn't use Solidity's built-in unit conversion
- Option D (10^12) doesn't represent any standard Ethereum unit

Understanding these conversions is crucial for handling financial transactions accurately in smart contracts.

Quiz 2: Using Ether Units in Contracts

Instructions: Neri is developing a barangay payment system. Which code snippet correctly sets up transaction fees using appropriate Ether units?

```
pragma solidity ^0.8.0;

contract BarangayPayments {
    // Option A
    uint256 public idCardFee = 0.001;
    uint256 public permitFee = 0.05;

    // Option B
    uint256 public idCardFee2 = 0.001 ether;
    uint256 public permitFee2 = 0.05 ether;

    // Option C
    uint256 public idCardFee3 = 1000000 wei;
    uint256 public permitFee3 = 50000000 wei;

    // Option D
    uint256 public idCardFee4 = 1000000;
    uint256 public permitFee4 = 50000000;
}
```

Which option correctly represents the fees using appropriate Ether units?

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

Answer: B) Option B

Explanation: Option B is the correct way to express Ether values in Solidity. It uses the built-in `ether` unit which automatically converts to Wei (the base unit) during compilation.

The other options have issues:

- Option A uses decimal values without specifying the unit, which will cause compilation errors as Solidity doesn't allow direct decimal literals
- Option C uses wei explicitly but requires manual calculation of the equivalent values
- Option D uses plain numbers without units, making it unclear what denomination they represent

Using explicit units like `ether` in Solidity code improves readability and prevents mistakes in value calculations.

Quiz 3: Understanding Gas Fees and Gwei

Instructions: Neri needs to understand gas fees to optimize her anti-Hackana smart contracts. Which statement about gas and Gwei is correct?

```
pragma solidity ^0.8.0;
```

```
contract GasExample {
    // Gas price examples in different units
    uint256 public gasPrice1 = 20 gwei;
    uint256 public gasPrice2 = 0.00000002 ether;
    uint256 public gasPrice3 = 20000000000 wei;
    uint256 public gasPrice4 = 20 * 10**9;
}
```

Which of the following statements is TRUE?

- A) `gasPrice1` and `gasPrice2` are equal
- B) `gasPrice1` and `gasPrice3` are equal
- C) `gasPrice1`, `gasPrice3`, and `gasPrice4` are all equal
- D) All four gas price variables are equal

Answer: C) `gasPrice1`, `gasPrice3`, and `gasPrice4` are all equal

Explanation: The correct answer is C because:

- `gasPrice1` = 20 gwei = 20 * 10^9 wei
- `gasPrice3` = 20000000000 wei = 20 * 10^9 wei
- `gasPrice4` = 20 _ 10^9 = 20 _ 10^9 wei

However, `gasPrice2` = 0.00000002 ether = 20 _ 10^9 wei / 10^18 wei per ether = 0.02 _ 10^9 wei, which is not equal to the others.

Understanding these conversions is important for estimating transaction costs and setting appropriate gas prices in the Ethereum network.

Quiz 4: Practical Ether Calculations

Instructions: Neri is auditing a payment contract used by the barangay. Help her identify which function correctly handles Ether values.

```
pragma solidity ^0.8.0;

contract PaymentHandler {
    // Option A
    function calculateTotal1(uint256 itemCount) public pure returns (uint256) {
        uint256 pricePerItem = 0.01 ether;
        return itemCount * pricePerItem;
    }

    // Option B
    function calculateTotal2(uint256 itemCount) public pure returns (uint256) {
        uint256 pricePerItem = 10000000000000000; // 0.01 ETH in wei
        return itemCount * pricePerItem;
    }

    // Option C
    function calculateTotal3(uint256 itemCount) public pure returns (uint256) {
```

```
        uint256 pricePerItem = 0.01;
        return itemCount * pricePerItem * 1 ether;
    }

    // Option D
    function calculateTotal4(uint256 itemCount) public pure returns (uint256) {
        return itemCount * 0.01;
    }
}
```

Which function correctly calculates the total cost for multiple items at 0.01 ETH each?

- A) calculateTotal1()
- B) calculateTotal2()
- C) calculateTotal3()
- D) calculateTotal4()

Answer: A) calculateTotal1()

Explanation: Function `calculateTotal1()` correctly uses Solidity's built-in `ether` unit for clearer, less error-prone code.

- Function A uses `0.01 ether` which correctly converts to Wei at compile time
- Function B uses the raw Wei value (which is correct but less readable)
- Function C tries to use a decimal number (0.01) which isn't allowed in Solidity without being a string
- Function D attempts to use a decimal without any unit specification, which will cause a compilation error

Using the built-in units like `ether` and `gwei` makes your code more readable and less prone to calculation errors when dealing with cryptocurrency values.