**Blockchain Study Notes Day 12:**

**Module 2 - Solidity Basics**  
**Chapter 8 - Loops in Solidity**

**Introduction to Loops**

Loops in Solidity are used to execute a block of code repeatedly, based on a specified condition. They help in automating repetitive tasks, such as iterating over arrays or performing calculations.

**1. Types of Loops in Solidity**

**1.1. for Loop**

* Executes a block of code a specific number of times.
* **Syntax**:

for (initialization; condition; increment) {

// Code to execute

}

* **Example**:

function sumNumbers(uint n) public pure returns (uint) {

uint sum = 0;

for (uint i = 1; i <= n; i++) {

sum += i;

}

return sum;

}

**1.2. while Loop**

* Executes a block of code as long as the specified condition is true.
* **Syntax**:

while (condition) {

// Code to execute

}

* **Example**:

function findFactorial(uint n) public pure returns (uint) {

uint result = 1;

uint i = n;

while (i > 1) {

result \*= i;

i--;

}

return result;

}

**1.3. do...while Loop**

* Executes a block of code once, and then repeats the execution as long as the condition is true.
* **Syntax**:

do {

// Code to execute

} while (condition);

* **Example**:

function countDown(uint n) public pure returns (uint) {

uint count = 0;

do {

count++;

n--;

} while (n > 0);

return count;

}

**2. Example Program Using Loops (Using Munawar)**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract MunawarLoops {

// Function to calculate the sum of an array using a `for` loop

function sumArray(uint[] memory numbers) public pure returns (uint) {

uint sum = 0;

for (uint i = 0; i < numbers.length; i++) {

sum += numbers[i];

}

return sum;

}

// Function to find the first even number in an array using a `while` loop

function findFirstEven(uint[] memory numbers) public pure returns (uint) {

uint i = 0;

while (i < numbers.length) {

if (numbers[i] % 2 == 0) {

return numbers[i];

}

i++;

}

return 0; // Return 0 if no even number is found

}

// Function to demonstrate a `do...while` loop

function demoDoWhile(uint n) public pure returns (uint) {

uint count = 0;

do {

count++;

n--;

} while (n > 0);

return count;

}

}

**3. Best Practices for Using Loops**

* **Gas Optimization**:
  + Avoid loops that run indefinitely or over large datasets, as they consume more gas.
* **Break Conditions**:
  + Use break to exit loops early when a specific condition is met.

for (uint i = 0; i < numbers.length; i++) {

if (numbers[i] == target) {

break;

}

}

* **Use with Caution**:
  + Be mindful of potential infinite loops that could lead to contract failures.

**Home Task**

1. **Modify the Example Program**:
   * Add a for loop function to find the maximum number in an array.
2. **Create a New Contract**:
   * Implement a contract with functions using all three loop types to perform different tasks.
3. **Experiment with Break and Continue**:
   * Write a function using for or while loops that uses continue to skip certain iterations.

**Conclusion**

Loops in Solidity provide a way to perform repetitive tasks efficiently. By mastering for, while, and do...while loops, developers can handle various computational requirements in their smart contracts. However, loops should be used judiciously to optimize gas consumption and avoid performance bottlenecks.

Day 12 Notes

***Prepared by Munawar Johar***