**Unit 4 – Blockchain and Decentralized Applications**

**Lab 3 Manual**

**Objective:**

By the end of this lab, students will:

* Understand and implement control structures in Solidity (e.g., conditionals, loops).
* Define and use functions in Solidity, including the concept of pure functions.
* Build, test, and deploy small Solidity contracts with Truffle and Ganache.
* Analyze and interpret contract behavior in a simulated environment.

Control structures are essential for decision-making and repeating actions. We'll cover two main types:

1. Conditionals: if, else if, else
2. Loops: for, while

We will demonstrate both conditionals and loops to calculate a simple grade and iterate through a loop to count.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract ControlStructures {

// This function checks if a number is even or odd

function isEven(uint number) public pure returns (string memory) {

if (number % 2 == 0) {

return "Even";

} else {

return "Odd";

}

}

// This function calculates the sum of numbers from 1 to n

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

uint sum = 0;

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

sum += i;

}

return sum;

}

// This function counts down from n to 0

function countdown(uint n) public pure returns (string memory) {

string memory result = "";

while (n > 0) {

result = string(abi.encodePacked(result, uintToString(n), " "));

n--;

}

return result;

}

// Helper function to convert uint to string

function uintToString(uint v) internal pure returns (string memory) {

if (v == 0) {

return "0";

}

uint maxLen = 78;

bytes memory reversed = new bytes(maxLen);

uint len = 0;

while (v != 0) {

uint remainder = v % 10;

v = v / 10;

reversed[len++] = bytes1(uint8(48 + remainder));

}

bytes memory s = new bytes(len);

for (uint i = 0; i < len; i++) {

s[i] = reversed[len - 1 - i];

}

return string(s);

}

}

Functions are crucial for creating modular and reusable code. They can be:

1. **Public**: Can be called by anyone.
2. **Internal**: Only callable within the contract.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SimpleFunctions {

uint public count = 0;

// Public function that increments the count

function incrementCount() public {

count += 1;

}

// Internal function that doubles the count

function doubleCount() internal {

count \*= 2;

}

// Public function to get the current count

function getCount() public view returns (uint) {

return count;

}

// Function to reset count based on a condition

function resetIfGreaterThan(uint threshold) public {

if (count > threshold) {

count = 0;

}

}

}

Pure functions are restricted in Solidity, making them gas-efficient because they do not read or modify the blockchain state. Lets see pure functions with basic arithmetic operations.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Arithmetic {

// Pure function: adds two numbers

function add(uint a, uint b) public pure returns (uint) {

return a + b;

}

// Pure function: subtracts two numbers

function subtract(uint a, uint b) public pure returns (uint) {

require(a >= b, "a should be greater than or equal to b");

return a - b;

}

// Pure function: multiplies two numbers

function multiply(uint a, uint b) public pure returns (uint) {

return a \* b;

}

}

**Run the programs above, and share your output.**

---------------------------------------------------------------------------------------------

**Some additional programs to try:**

//SPD-License-Identifier: GPL-3.0

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

//how to write a contract

contract test{

//these all are state variable, state variable identify

//if we show the variable after execution the we use as public

    uint256 public a1;

    bool public abc;

    string public xyz;

    address public owner=msg.sender;

    //global variables send data on the blockchain

constructor(){

}

//function declaration but this will be write function

function testing(uint a, uint b ) public {

   a1=a+b;

}

function contractvalue() public view  returns(address)  {

//if we  return address(this), then it will return contract address which will be change everytime if we execute

return msg.sender; //this will return the account address of this remix

}

function timecheck() public view returns(uint){// view means that this read only function

//the output value 1730997472 is check on the below site

//https://www.epochconverter.com/

return block.timestamp;

}

}

//SPD-License-Identifier: GPL-3.0

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract functiontype{

//visibility indentifier is view,simple, pure

uint public  statevariable; //if we put public with this variable then viewfunction() is not required

//any function which send data on blockchain is said to be setter function or write function,

function simpleFunction(uint a) public {

    statevariable=a;

}

///////////////////////////////////////

//readonly function or like getter function

//view fintion used to take data from blockchain

// function viewfunction() public view returns(uint){// if not declare the statevariable as public then this function used

// return statevariable;

// }

//////////////////////////////////////

//pure function is sued for add an additional functionality which return the data

 function pureFunction(uint a, uint b) public pure returns(uint) {

   uint c=a+b;

   return c;

 }

function callPure(uint a, uint b) public {

  statevariable=pureFunction(a,b);

}

}