# PRACTICAL:1

### **A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.**

import binascii

import Crypto

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

def init (self):

random = Crypto.Random.new().read self.\_private\_key = RSA.generate(1024, random) self.\_public\_key = self.\_private\_key.publickey() self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

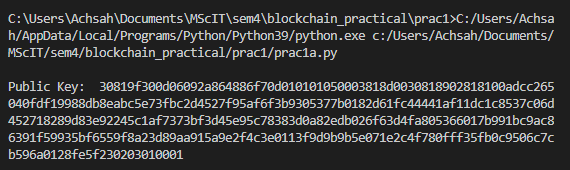
@property

def identity(self): return

binascii.hexlify(self.\_public\_key.exportKey(format="DER")).decode( "ascii"

)

Dinesh = Client()

print("\n Public Key:",Dinesh.identity)

**Output:**

### **A transaction class to send and receive money and test it.**

import binascii import collections import datetime

from client import Client from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Transaction:

def init (self, sender, recipient, value): self.sender = sender

self.recipient = recipient self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

identity = "Genesis" if self.sender = "Genesis" else self.sender.identity

return collections.OrderedDict(

{

"sender": identity, "recipient": self.recipient, "value": self.value, "time": self.time,

}

)

def sign\_transaction(self):

private\_key = self.sender.\_private\_key signer = PKCS1\_v1\_5.new(private\_key)

h = SHA.new(str(self.to\_dict()).encode("utf8"))

return binascii.hexlify(signer.sign(h)).decode("ascii")

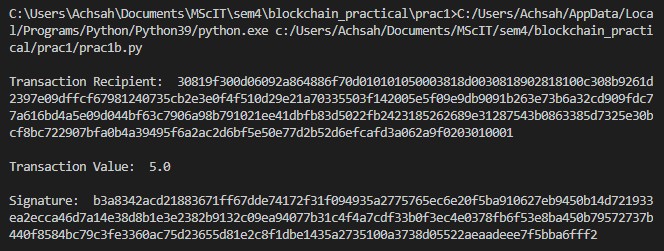
Dinesh = Client() Ramesh = Client()

t = Transaction(Dinesh, Ramesh.identity, 5.0) print("\nTransaction Recipient:\n", t.recipient) # print("\nTransaction Sender:\n", t.sender)

print("\nTransaction Value:\n", t.value)

signature = t.sign\_transaction() print("\nSignature:\n", signature)

**Output:**



### **Create multiple transactions and display them.**

from client import Client

from transaction\_class import Transaction

Dinesh = Client() Ramesh = Client()

t = Transaction(Dinesh, Ramesh.identity, 5.0) print("\nTransaction Recipient:\n", t.recipient) # print("\nTransaction Sender:\n", t.sender) print("\nTransaction Value:\n", t.value)

signature = t.sign\_transaction() print("\nSignature:\n", signature)

Dinesh = Client() Ramesh = Client() Seema = Client() Vijay = Client()

t1 = Transaction(Dinesh, Ramesh.identity, 15.0) t1.sign\_transaction()

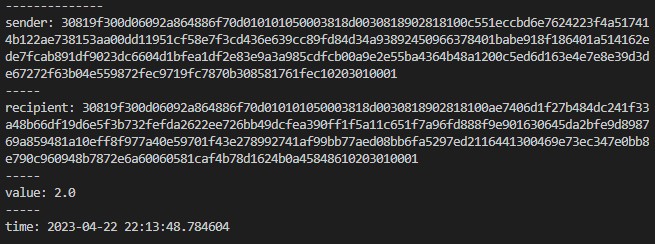
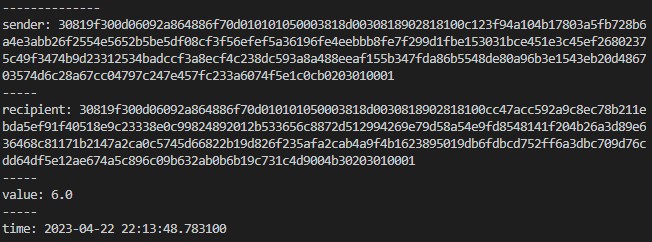
transactions = [t1]

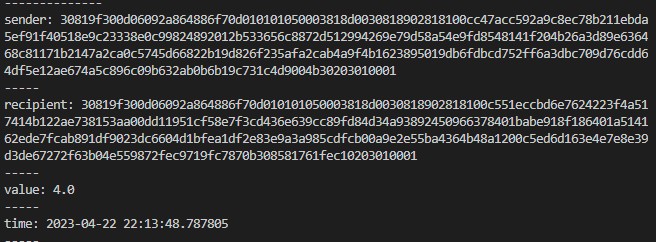
t2 = Transaction(Dinesh, Seema.identity, 6.0) t2.sign\_transaction() transactions.append(t2)

t3 = Transaction(Ramesh, Vijay.identity, 2.0) t3.sign\_transaction() transactions.append(t3)

t4 = Transaction(Seema, Ramesh.identity, 4.0) t4.sign\_transaction() transactions.append(t4)

for transaction in transactions: Transaction.display\_transaction(transaction) print("–————————————–")

**Output:**



### **Create a blockchain, a genesis block and execute it.**

from client import Client

from transaction\_class import Transaction

class Block:

def init (self, client): self.verified\_transactions = [] self.previous\_block\_hash = "" self.Nonce = ""

self.client = client

def dump\_blockchain(blocks):

print(f"\nNumber of blocks in the chain: {len(blocks)}")

for i, block in enumerate(blocks): print(f"block # {i}")

for transaction in block.verified\_transactions: Transaction.display\_transaction(transaction) print("–————————————–")

print("=====================================")

Dinesh = Client()

t0 = Transaction("Genesis", Dinesh.identity(), 500.0)

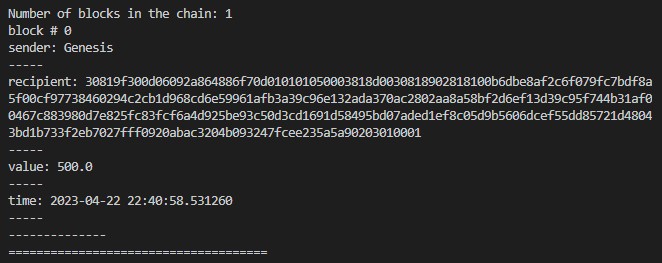
block0 = Block(Dinesh) block0.previous\_block\_hash = ""

NONCE = None

block0.verified\_transactions.append(t0) digest = hash(block0)

last\_block\_hash = digest

TPCoins = [block0] dump\_blockchain(TPCoins)



### **Create a mining function and test it.**

import hashlib

def sha256(message):

return hashlib.sha256(message.encode("ascii")).hexdigest()

def mine(message, difficulty=1): assert difficulty ÷ 1 prefix = "1" \* difficulty for i in range(1000):

digest = sha256(str(hash(message)) + str(i)) if digest.startswith(prefix):

print(f"after {str(i)} iterations found nonce: {digest}") # return print(digest)

mine("test message", 2)

**Output:**



**F] Add blocks to the miner and dump the blockchain.**

import datetime import hashlib

# Create a class with two functions

class Block:

def init (self, data, previous\_hash):

self.timestamp = datetime.datetime.now(datetime.timezone.utc) self.data = data

self.previous\_hash = previous\_hash self.hash = self.calc\_hash()

def calc\_hash(self):

sha = hashlib.sha256()

hash\_str = self.data.encode("utf-8") sha.update(hash\_str)

return sha.hexdigest()

# Instantiate the class

blockchain = [Block("First block", "0")]

blockchain.append(Block("Second block", blockchain[0].hash)) blockchain.append(Block("Third block", blockchain[1].hash))

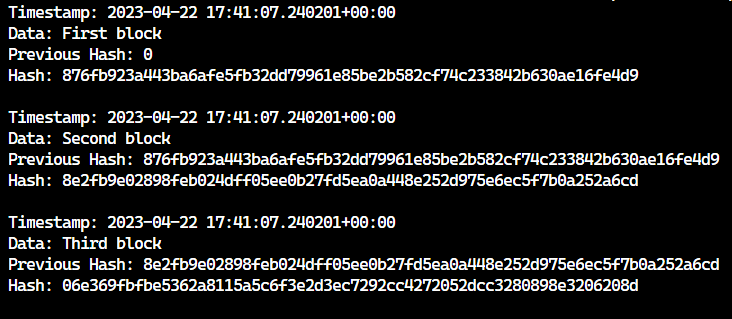
# Dumping the blockchain

for block in blockchain: print(

f"Timestamp: {block.timestamp}\nData: {block.data}\nPrevious Hash:

{block.previous\_hash}\nHash: {block.hash}\n")

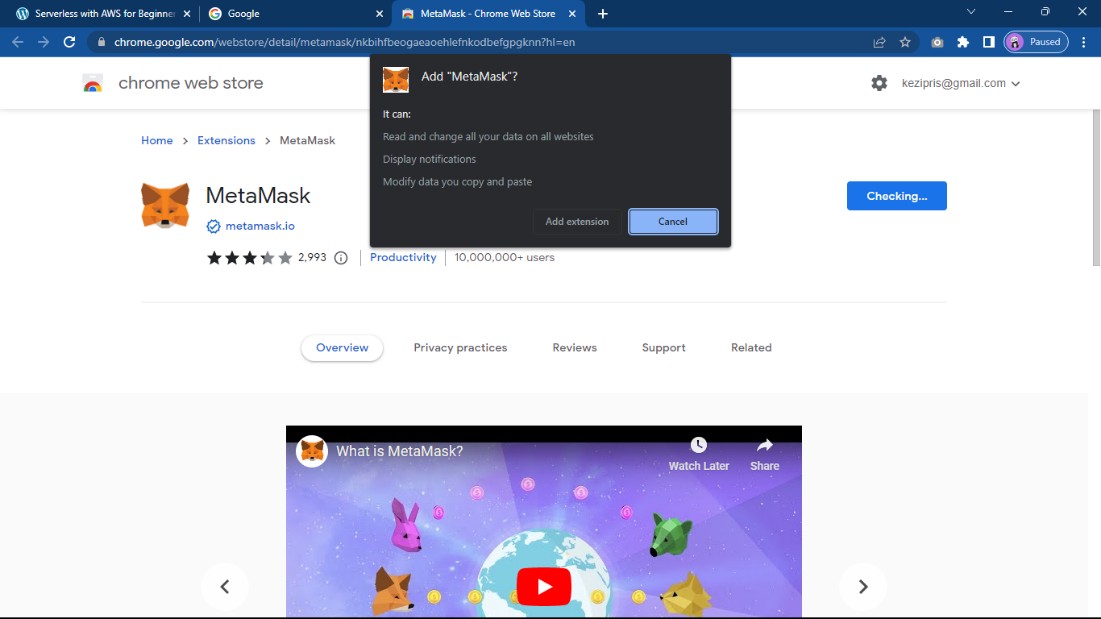
**Output:**

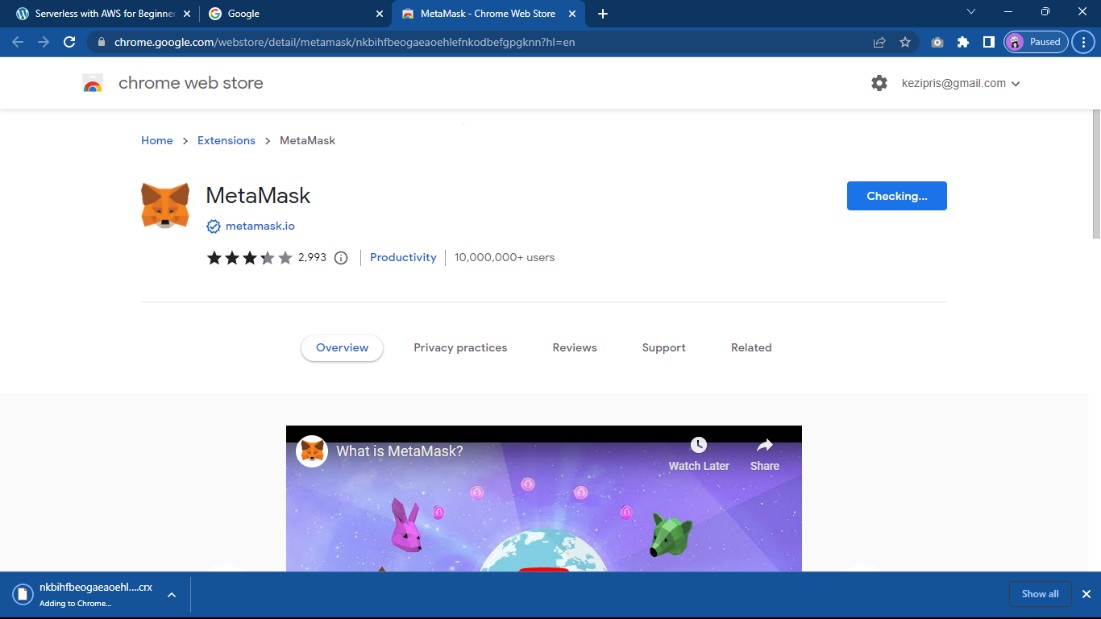


# PRACTICAL-2

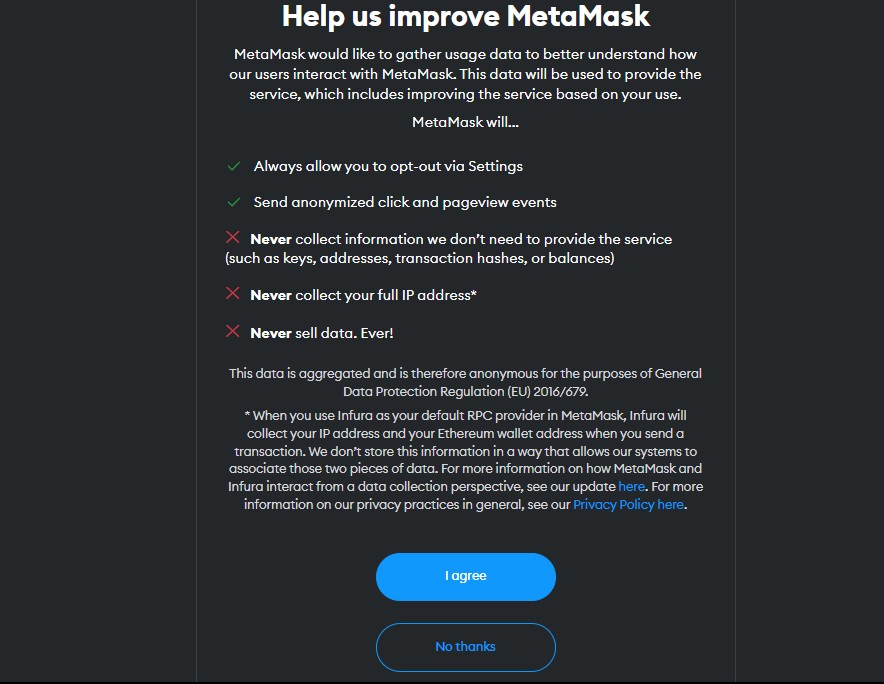
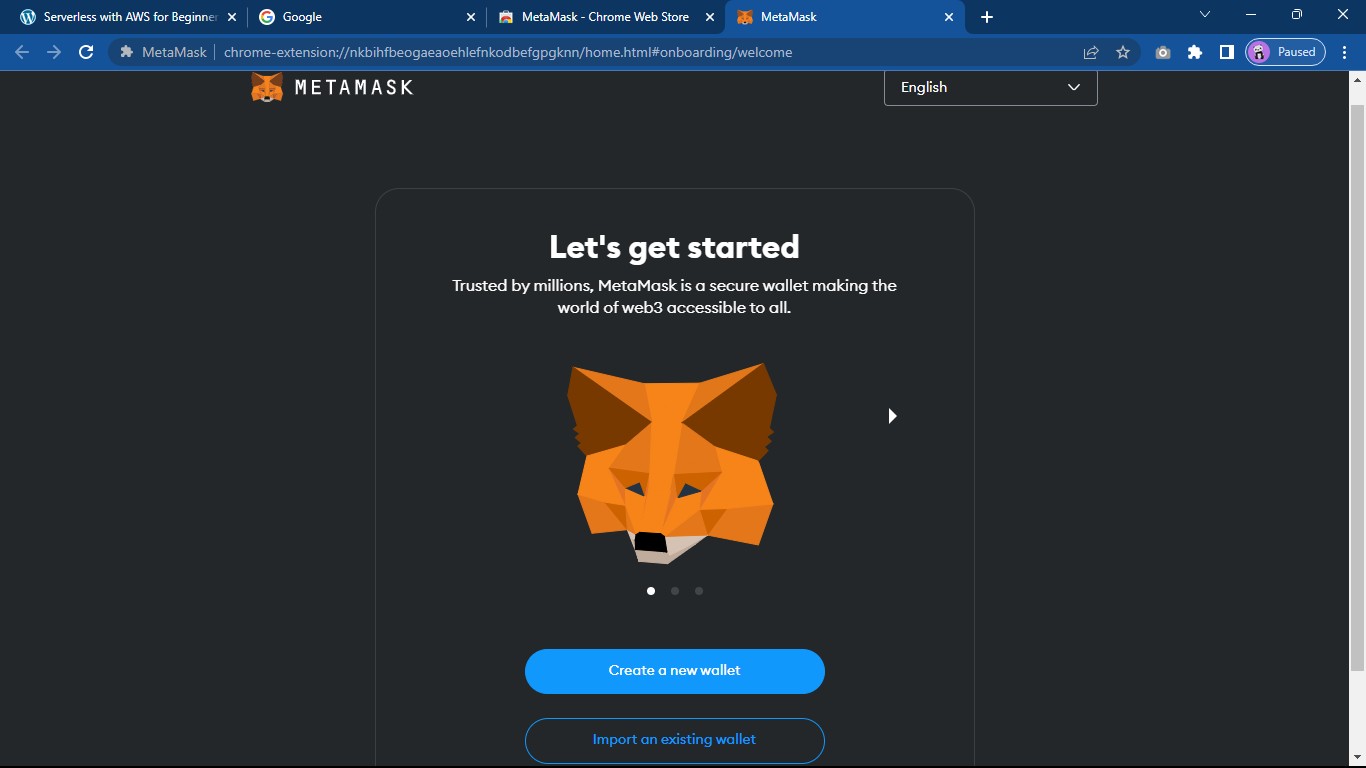
# Aim: Install and configure go Ethereum and the mist browser. develop and test a sample application (MetaMask & remix)

## **Step 1->** Install MetaMask extension for chrome from Chrome Web Store

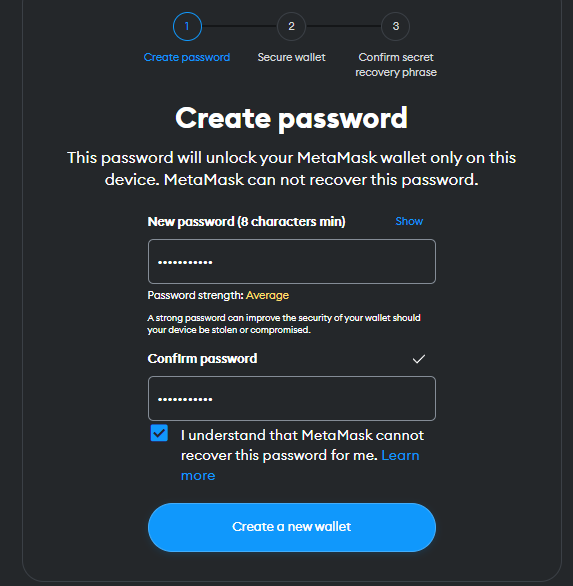


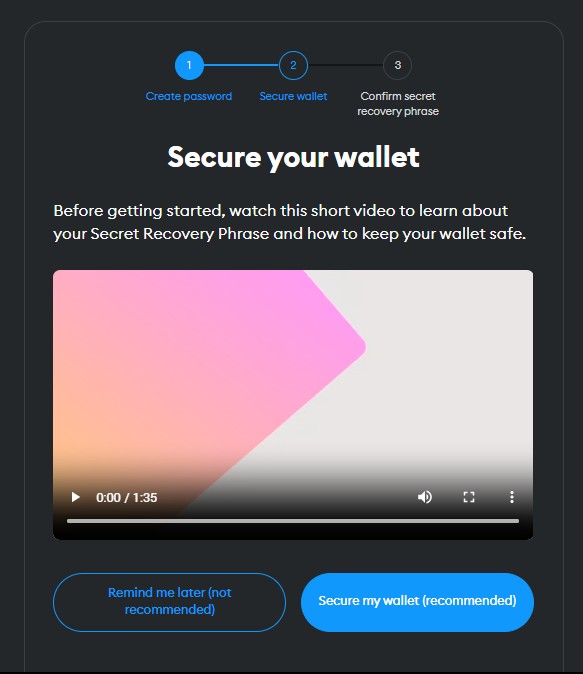


**Step 2->** Click on Metamask Extension in Extensions. Below page will open in a new tab. Click on Create a New Wallet. Click on I agree.

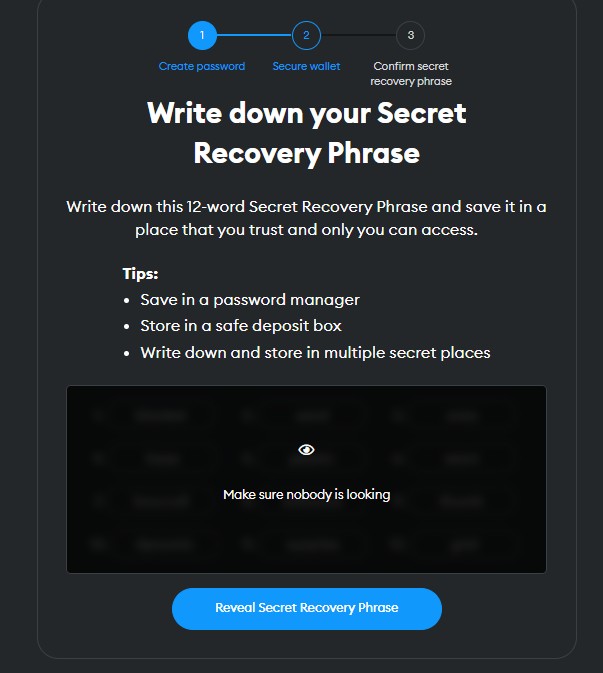


## **Step 3->** Create a password. This password can be used only on the device it was created on. Create a Strong password and click on Create a new Wallet button

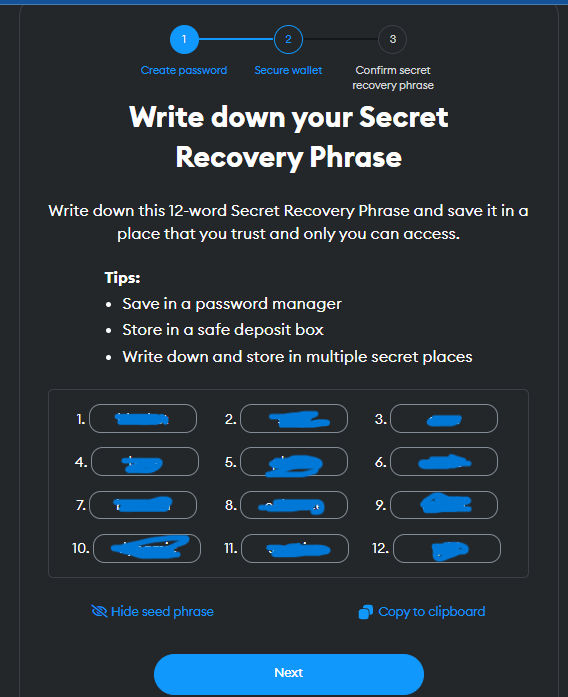




**Step 4->** Click on Secure my wallet button, following window will appear



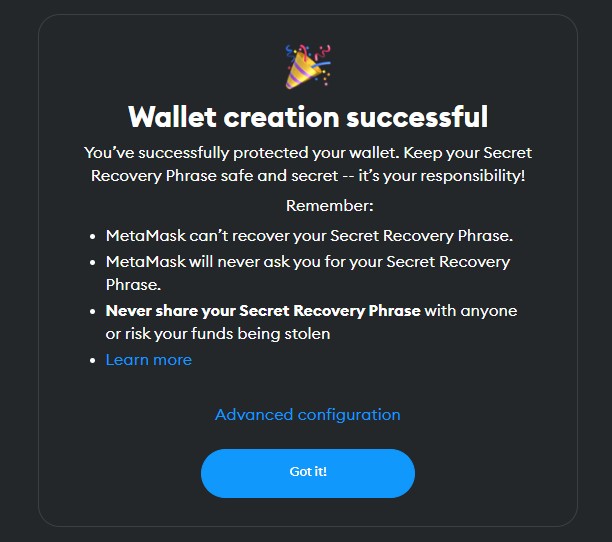
## **Step 5->** Click on Reveal Secret Recovery Phrase button and save the words in the same sequence



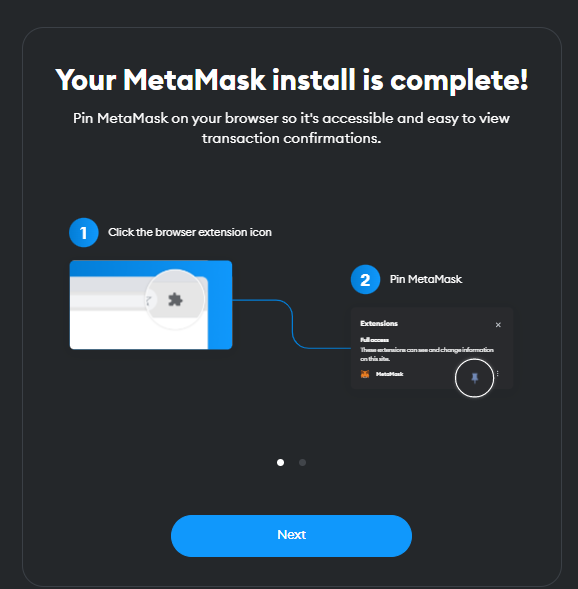
**Step 6->** Enter the respective words in the empty positions and click Confirm.



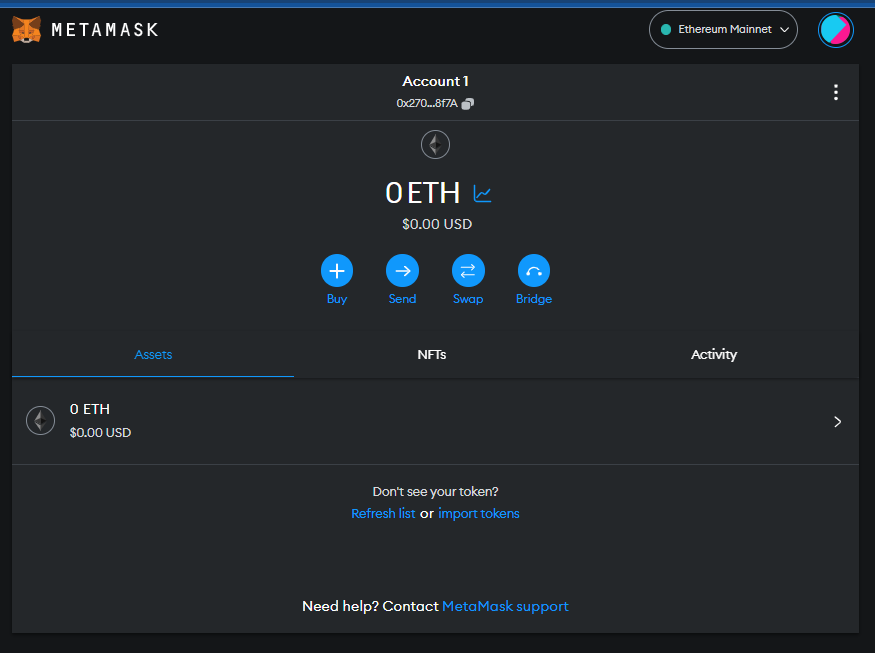
**Step 7->** Click Got it!



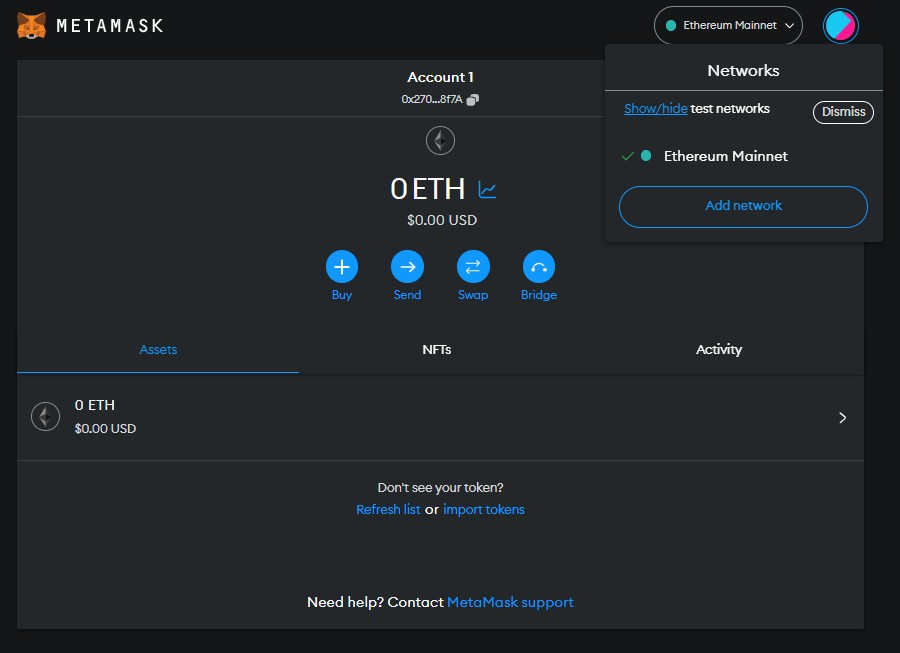
**Step 8->** Click on Next

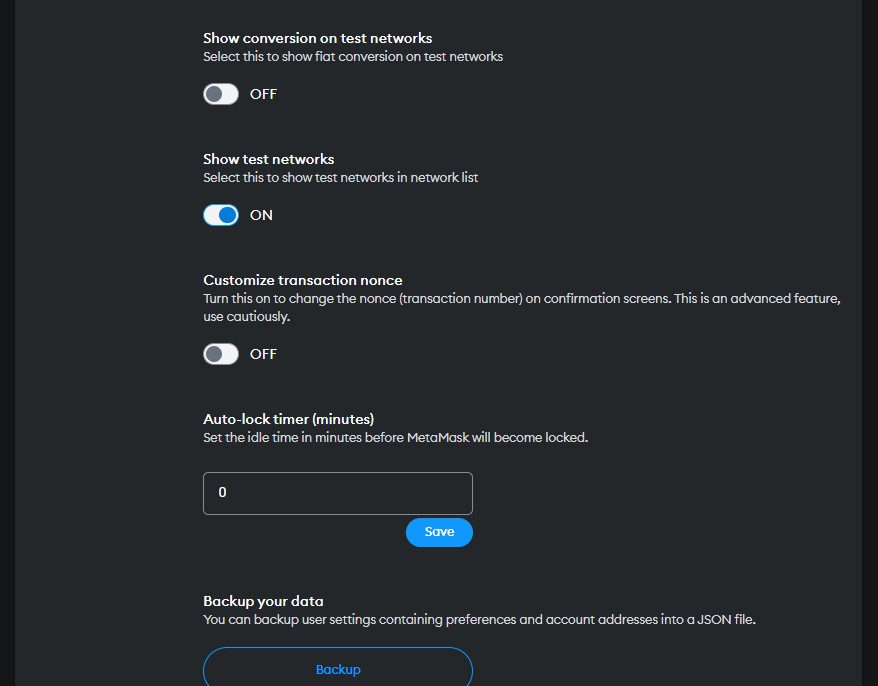


**Step 9->** Following will be the Dashboard

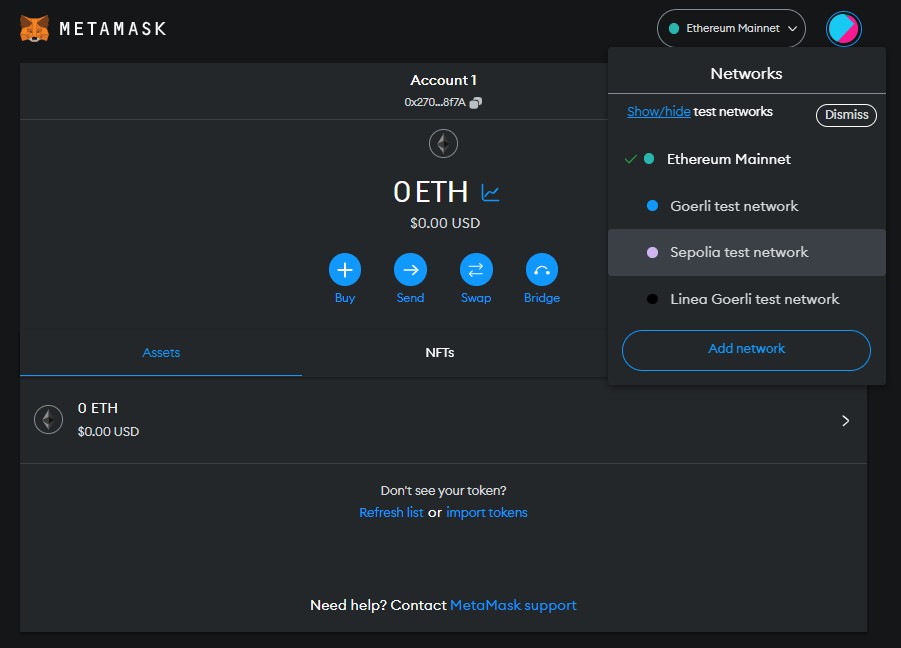


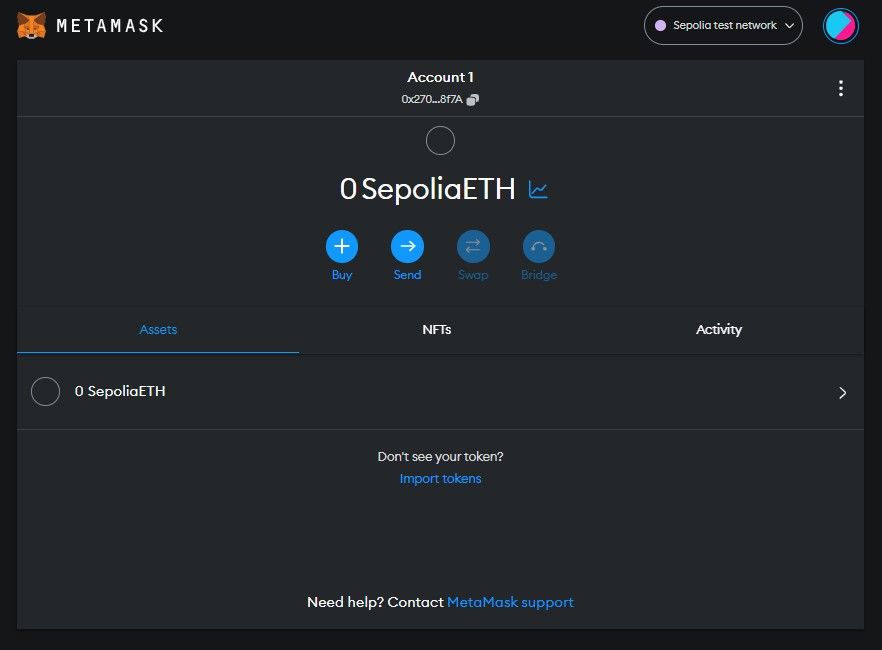
## **Step 10->** Click on Ethereum Mainnet button. Next click on Show/hide test networks.



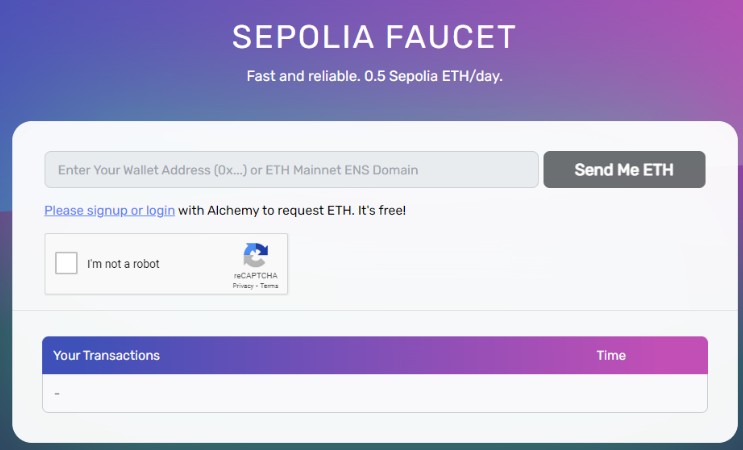


**Step 11->** Check if tesnets are shown by clicking on Etherum Mainnet button. Click on Sepolia test network.

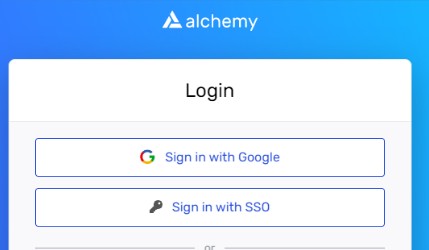




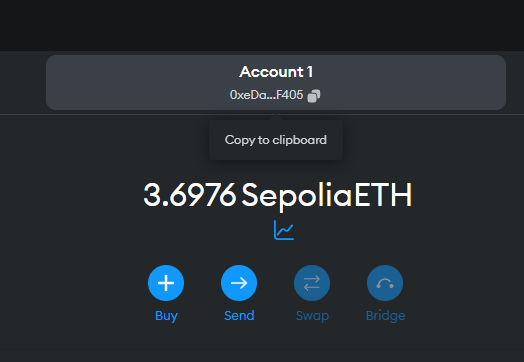
**Step 12->** Go to <https://sepoliafaucet.com/>and Click on Alchemy Login button.



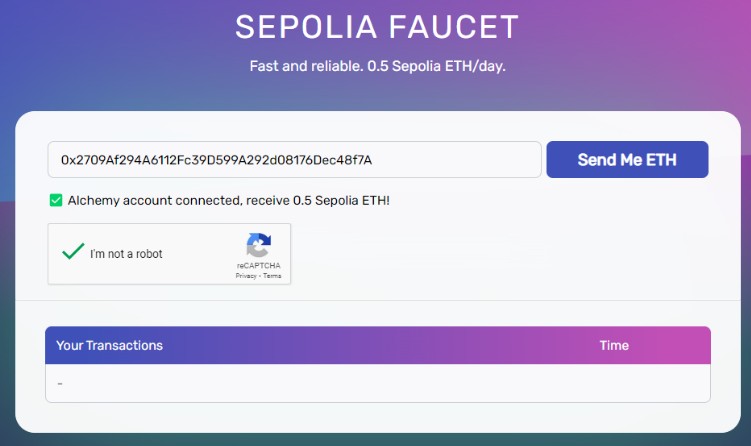
**Step 13->** Login to a gmail account in another browser tab and click on Sign in with Google



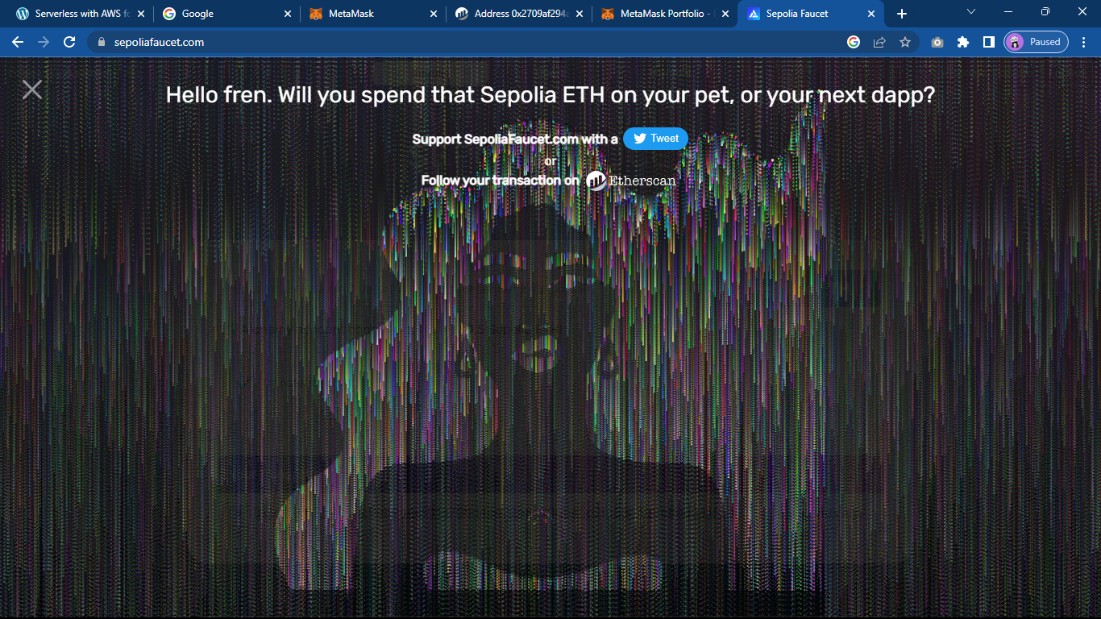
## **Step 14->** Now go to MetaMask and copy the account address.



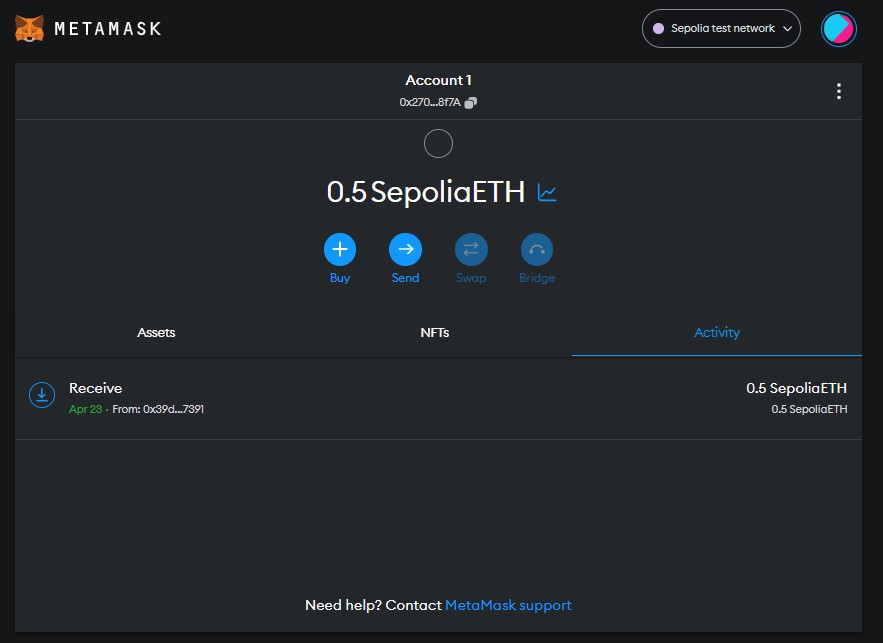
**Step 15->** Paste the address and click on Send Me ETH.



## **Step 16->** Your ETH transfer is succesfull. You should see a similar animation.



**Step 17->** Check your MetaMask account for Sepolia test network. 0.5 ETH will be added.



# PRACTICAL-3

# Aim: Implement and demonstrate the use of the following in solidity\

1. TO EXECUTE SOLIDITY SCRIPTS GO TO ->[HTTPS://REMIX.ETHEREUM.ORG/](https://remix.ethereum.org/)
2. OPEN CONTRACTS FOLDER AND STARTING WRITING SCRIPTS. THE SCRIPTS ARE COMPILED USING SOLIDITY COMPILER.
3. THE FOLLOWING SCRIPTS WERE COMPILED USING 0.5.0+COMMIT.1D4F565A SOLIDITY COMPILER
4. DEPLOY THE SCRIPTS TO EXECUTE CODE

### **Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables**

# Variable

pragma solidity ^0.5.0;

contract variable\_demo {

uint256 sum = 4; //state variable uint256 x;

address a;

string s = "welcome";

function add(uint256) public {

uint256 y = 2; //local variable sum = sum+x+y: sum = sum + x + y;

}

function display() public view returns (uint256) { return sum;

}

function displayMsg() public view returns (string memory) { return s;

}

}

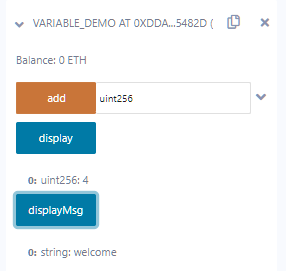
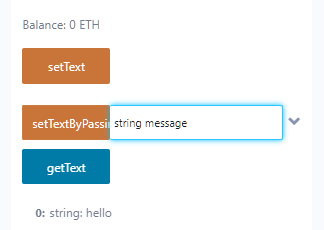


Figure 1 -Displaying variable value

# Strings



pragma solidity ^0.5.0;

contract LearningStrings { string text;

function getText() public view returns (string memory) { return text;

}

function setText() public { text = "hello";

}

function setTextByPassing(string memory message) public { text = message;

}

}

Figure 2 - Before setting new string value

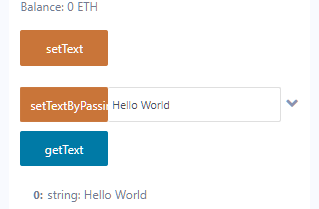
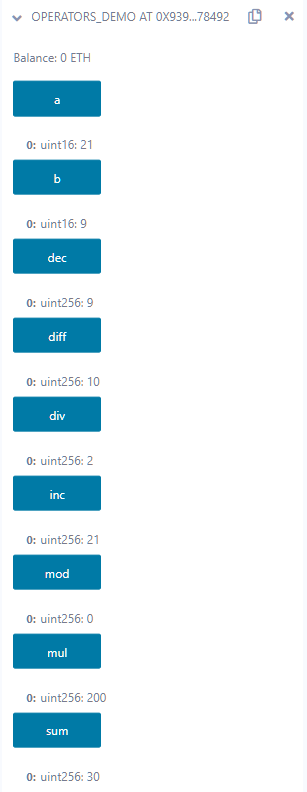


Figure 3 - After setting string value

# Operators



pragma solidity ^0.5.0;

contract SolidityTest {

uint16 public a = 20; uint16 public b = 10;

uint256 public sum = a + b; uint256 public diff = a - b; uint256 public mul = a \* b; uint256 public div = a / b; uint256 public mod = a % b; uint256 public dec = --b;

uint256 public inc = ++a;

}

Figure 4 - All operators of solidity displayed

# Array

pragma solidity ^0.5.0; contract arraydemo

{

//Static Array

uint[6] arr2=[10,20,30];

function dispstaticarray() public view returns(uint[6] memory)

{

return arr2;

}

//Dynamic Array uint x=5;

uint [] arr1;

function arrayDemo() public

{

while(x>0)

{

arr1.push(x); x=x-1;

}

}

function dispdynamicarray() public view returns(uint[] memory)

{

return arr1;

}

}

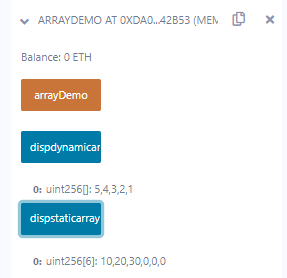


Figure 5 - Array displayed

# Decision Making

If Else

pragma solidity ^0.5.0; contract ifelsedemo

{

uint i=10;

function decision\_making() public view returns(string memory)

{

if(i%2==0)

{

return "even";

}

else

{

return "Odd";

}

}

}



Figure 6 - If else output

# Loops

pragma solidity ^0.5.0; contract loopDemo

{

uint [] data;

## For Loop

function forDemo() public returns(uint[] memory)

{

for(uint i=0; i<10; i++){ data.push(i);

}

return data;

}

function disp() public view returns(uint[] memory)

{

return data;

}

}

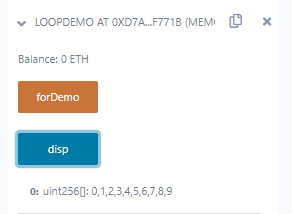


Figure 7 - Appending values to array using for loop

While

Loop

pragma solidity ^0.5.0; contract whiledemo

{

uint [] data; uint x=0;

function whileLoopDemo() public

{

while(x<5)

{

data.push(x); x=x+1;

}

}

function dispwhileloop() public view returns(uint[] memory)

{

return data;

}

}

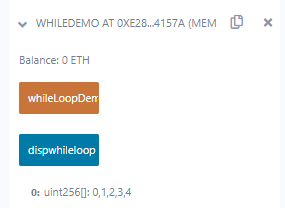


Figure 8 - Appending values to array using while loop

## Do

While

pragma solidity ^0.5.0;

// Creating a contract contract DoWhile {

// Declaring a dynamic array uint256[] data;

// Declaring state variable uint8 j = 0;

// Defining function to demonstrate

// 'Do-While loop'

function loop() public returns (uint256[] memory) { do {

j++;

data.push(j);

} while (j < 5); return data;

}

function display() public view returns(uint256[] memory){ return data;

}

}

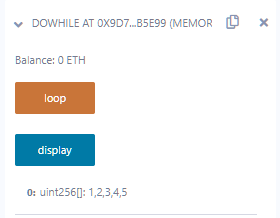
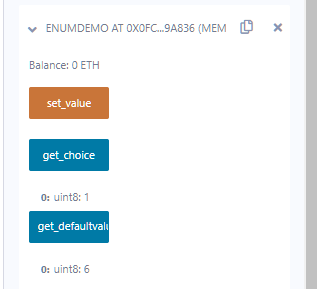


Figure 9 Appending values to array using do while loop

# Enums



pragma solidity ^0.5.0;

contract enumdemo { enum week\_days {

Monday, Tuesday,

Wednesday, Thursday, Friday,

Saturday, Sunday

}

week\_days week;

week\_days choice;

week\_days constant default\_value = week\_days.Sunday;

function set\_value() public {

choice = week\_days.Tuesday;

}

function get\_choice() public view returns (week\_days) { return choice;

}

function get\_defaultvalue() public view returns (week\_days) { return default\_value;

}

}

Figure 10 - Accessing enum values

# Structs

pragma solidity ^0.5.0;

contract structdemo { struct Book {

string name;

string author; uint256 id;

bool availability;

}

Book book2;

Book book1 = Book("A Little Life", "Hanya Yanagihara", 2, false);

function set\_details() public {

book2 = Book("Almond", "Sohn won-pyung", 1, true);

}

function book\_info() public

view

returns (

string memory, string memory, uint256,

bool

)

{

return (book1.name, book1.author, book1.id, book1.availability);

}

function get\_details() public

view

returns (

string memory, string memory, uint256, bool

)

{

return (book2.name, book2.author, book2.id, book2.availability);

}

}

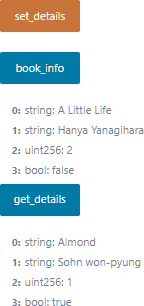
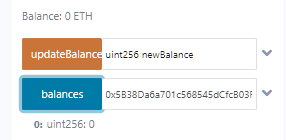


Figure 11- Structure datatype in solidity

# Mappings



pragma solidity ^0.5.0;

contract LedgerBalance {

mapping(address => uint256) public balances;

function updateBalance(uint256 newBalance) public { balances[msg.sender] = newBalance;

}

}

contract Updater {

function updateBalance() public returns (uint256) {

LedgerBalance ledgerBalance = new LedgerBalance(); return ledgerBalance.balances(address(this));

}

}

Figure 12 - Before updating balance

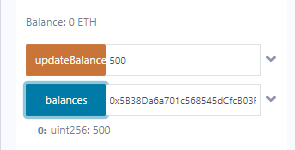


Figure 13 - After updating balance

# Conversions

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

contract ImplicitConversion {

function add() public pure returns (uint256) { uint256 a = 10;

uint256 b = 20; return a + b;

}

}

contract ExplicitConversion {

function convert() public pure returns (bytes memory) { string memory str = "Hello World";

bytes memory b = bytes(str); return b;

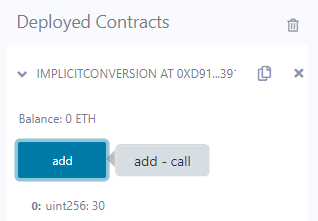
}

}

**Step 1->** Deploy both contracts



## **Step 2->** Open Implicit Conversion and click on add button to sum and display value



**Step 3->** Open Explicit Conversion and click on convert button

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

contract SolidityTest {

function convert\_Amount\_to\_Wei(uint256 Amount) public

pure

returns (uint256)

{

return Amount \* 1 wei;

}

function convert\_Amount\_To\_Ether(uint256 Amount) public

pure

returns (uint256)

{

return Amount \* 1 ether;

}

function convert\_Amount\_To\_Gwei(uint256 Amount) public

pure

returns (uint256)

{

return Amount \* 1 gwei;

}

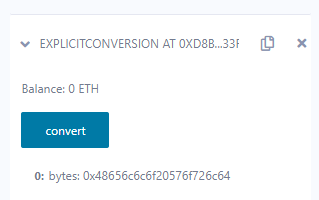
function convert\_seconds\_To\_mins(uint256 \_seconds) public

pure

returns (uint256)

{

return \_seconds / 60;



# Ether Units

}

function convert\_seconds\_To\_Hours(uint256 \_seconds) public

pure

returns (uint256)

{

return \_seconds / 3600;

}

function convert\_Mins\_To\_Seconds(uint256 \_mins) public

pure

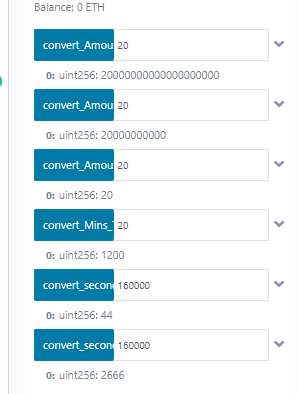
returns (uint256)

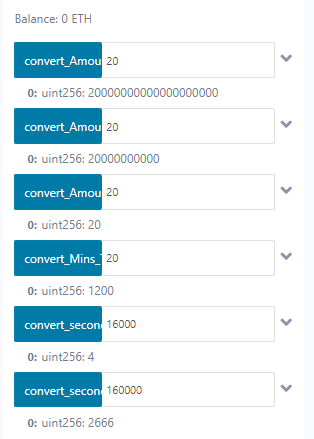
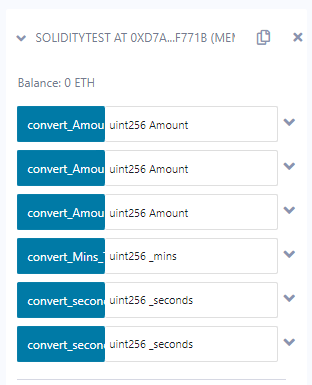
{

return \_mins \* 60;

}

}



**Step 1->** Provide values to each function and click on them

# Special Variables

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

contract Special\_Variables {

mapping(address => uint256) rollNo;

function setRollNO(uint256 \_myNumber) public { rollNo[msg.sender] = \_myNumber;

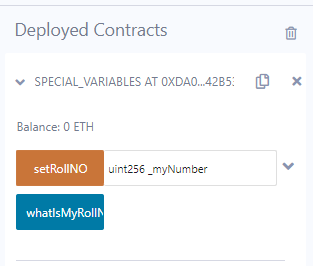
}

function whatIsMyRollNumber() public view returns (uint256) { return rollNo[msg.sender];

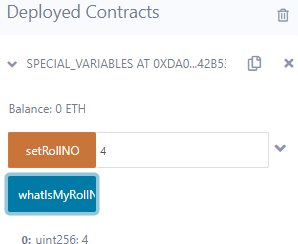
}

}

**Step 1->** Deploy contract Special Variables

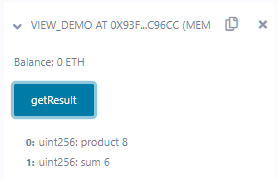


## **Step 2->** Input a number for setRollNO function and click on it & whatIsMyRollNumber button



1. **Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions**

# View Functions



pragma solidity ^0.5.0;

contract view\_demo { uint256 num1 = 2; uint256 num2 = 4;

function getResult() public view returns (uint256 product, uint256 sum) { product = num1 \* num2;

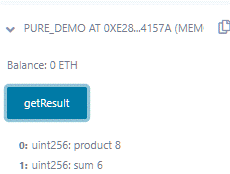
sum = num1 + num2;

}

}

Figure 14 - View function demo

# Pure Functions



pragma solidity ^0.5.0;

contract pure\_demo {

function getResult() public pure returns (uint256 product, uint256 sum) { uint256 num1 = 2;

uint256 num2 = 4;

product = num1 \* num2; sum = num1 + num2;

}

}

Figure 15 - Pure function output

# Mathematical Functions

pragma solidity ^0.5.0; contract Test{

function CallAddMod() public pure returns(uint){ return addmod(7,3,3);

}

function CallMulMod() public pure returns(uint){ return mulmod(7,3,3);

}

}

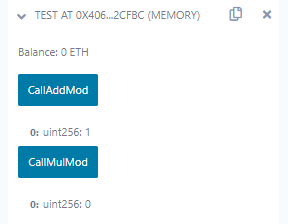


Figure 16 - Mathematical functions in solidity

# Cryptographic Functions

pragma solidity ^0.5.0; contract Test{

function callKeccak256() public pure returns(bytes32 result){ return keccak256("BLOCKCHAIN");

}

function callsha256() public pure returns(bytes32 result){ return sha256("BLOCKCHAIN");

}

function callripemd() public pure returns (bytes20 result){ return ripemd160("BLOCKCHAIN");

}

}

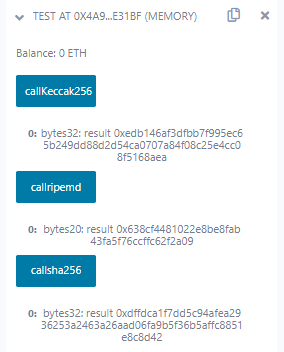


Figure 17 - Cryptography algorithms in solidity

# Functions

// SPDX-License-Identifier: MIT pragma solidity >=0.4.22 <0.9.0;

contract Test {

function return\_example() public

pure

returns (

uint256, uint256, uint256,

string memory

)

{

uint256 num1 = 10; uint256 num2 = 16;

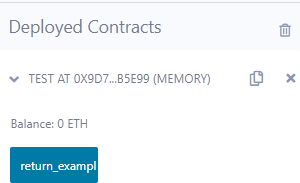
uint256 sum = num1 + num2; uint256 prod = num1 \* num2; uint256 diff = num2 - num1;

string memory message = "Multiple return values"; return (sum, prod, diff, message);

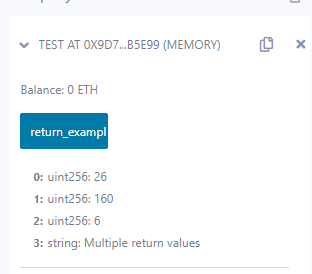
}

}

**Step 1->** Deploy Test Contract



**Step 2->** Click on return\_example button to display all values



# Fallback Function

// SPDX-License-Identifier: MIT pragma solidity ^0.5.12;

contract A {

uint256 n;

function set(uint256 value) external { n = value;

}

function() external payable { n = 0;

}

}

contract example {

function callA(A a) public returns (bool) {

(bool success, ) = address(a).call(abi.encodeWithSignature("setter()")); require(success);

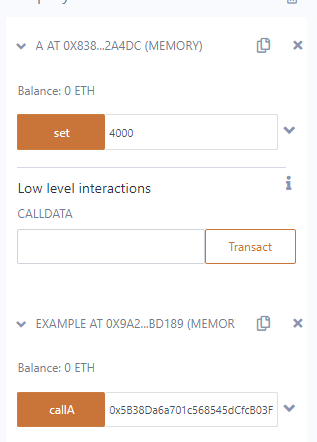
address payable payableA = address(uint160(address(a))); return (payableA.send(2 ether));

}

}

**Step 1->** Deploy both A & example contracts



**Step 2->** Provide values to both deployed contracts accordingly(use any address)

# Function Overloading

// SPDX-License-Identifier: MIT pragma solidity ^0.8.0;

contract OverloadingExample {

function add(uint256 a, uint256 b) public pure returns (uint256) { return a + b;

}

function add(string memory a, string memory b) public

pure

returns (string memory)

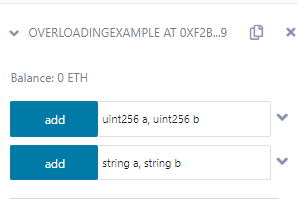
{

return string(abi.encodePacked(a, b));

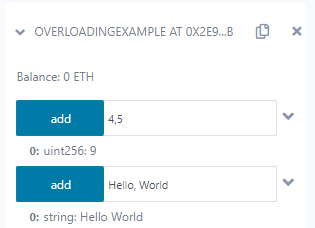
}

}

**Step 1->** Deploy Overloading Example contract



**Step 2->** Give integer and string values to both add functions as below



# Function modifiers

// SPDX-License-Identifier: MIT pragma solidity ^0.5.0;

contract ExampleContract {

address public owner = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4; uint256 public counter;

modifier onlyowner() {

require(msg.sender == owner, "Only the contract owner can call");

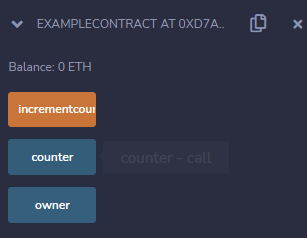
\_;

}

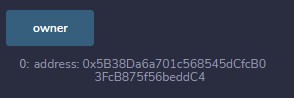
function incrementcounter() public onlyowner { counter++;

}

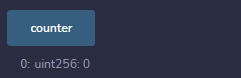
}



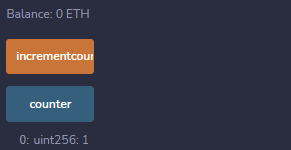
**Step 1->** Click on owner button



## **Step 2->** Click on counter button initially it is 0.



**Step 3->** Then click on increment counter button and again click on counter button, the counter has been increased



# PRACTICAL-4

# Aim: Implement and demonstrate the use of the following in solidity

## **Withdrawal Pattern, Restricted Access**

### Withdrawal Pattern

// SPDX-License-Identifier: MIT pragma solidity 0.8.18;

contract WithdrawalPattern { address public owner;

uint256 public lockedbalance;

uint256 public withdrawablebalance;

constructor() {

owner = msg.sender;

}

modifier onlyowner() {

require(msg.sender == owner, "Only the owner can call this function");

\_;

}

function deposit(uint256 amount) public payable {

require(amount > 0, "Amount must be greater than zero"); lockedbalance += amount;

}

function withdraw(uint256 amount) public payable onlyowner { require(

amount <= withdrawablebalance,

"Insufficient withdrawable balance"

);

withdrawablebalance -= amount;

payable(msg.sender).transfer(amount);

}

function unlock(uint256 amount) public onlyowner {

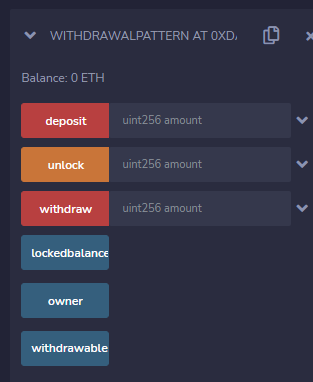
require(amount <= lockedbalance, "Insufficient locked balance"); lockedbalance -= amount;

withdrawablebalance += amount;

}

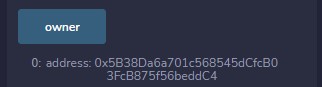
}

**Output:**

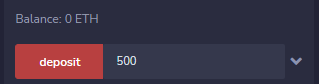


# Flow of execution

**Step 1->** Click on owner



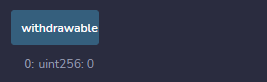
## **Step 2->** Enter an amount and click on deposit



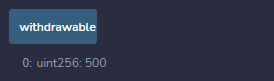
**Step 3->** Click on locked balance button to display the locked amount in the account



## **Step 4->** Click on withdrawable balance button

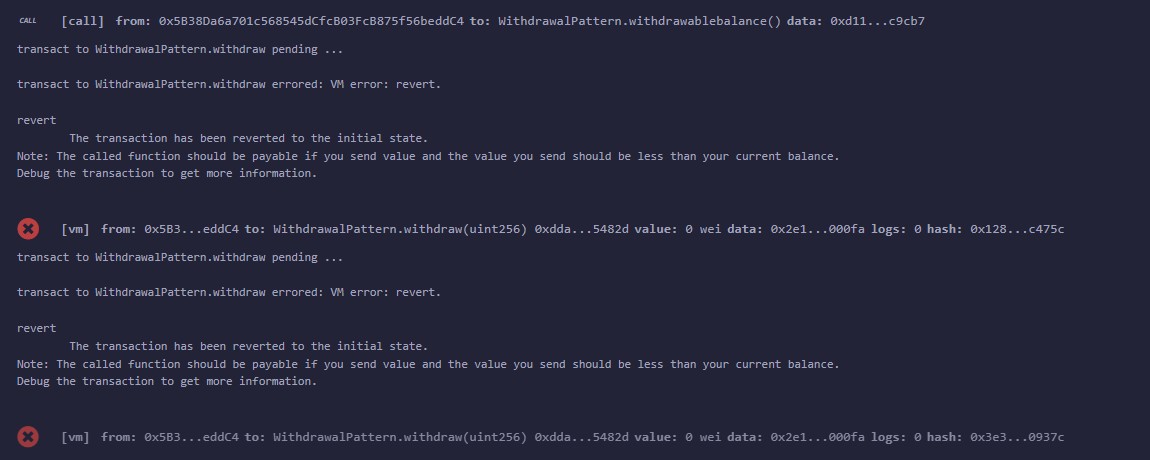


**Step 5->** Click on unlock button and enter any amount to transfer amount to withdrawable balance. Check locked balance and withdrawable balance.



## **Step 6->** Enter any amount you want to withdraw and Click the withdraw button.

You should get an error and the transaction should be reverted.



### Restricted Access

//SPDX-License-Identifier: MIT pragma solidity ^0.8.18;

contract RestrictedAccess {

address public owner = msg.sender;

uint256 public creationTime = block.timestamp;

modifier onlyBy(address \_account) {

require(msg.sender == \_account, "Sender not authorized!");

\_;

}

modifier onlyAfter(uint256 \_time) {

require(block.timestamp >= \_time, "Function was called too early!");

\_;

}

modifier costs(uint256 \_amount) {

require(msg.value >= \_amount, "Not enough Ether provided!");

\_;

}

function forceOwnerChange(address \_newOwner) public

payable

costs(200 ether)

{

owner = \_newOwner;

}

function changeOwner(address \_owner) public onlyBy(owner) { owner = \_owner;

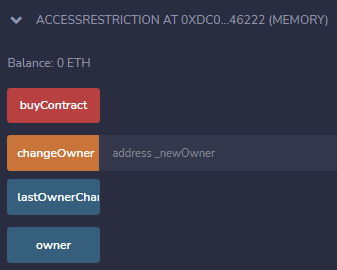
}

function disown() public onlyBy(owner) onlyAfter(creationTime + 3 weeks) { delete owner;

}

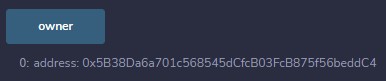
}

**Output:**

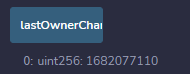


# Flow of execution

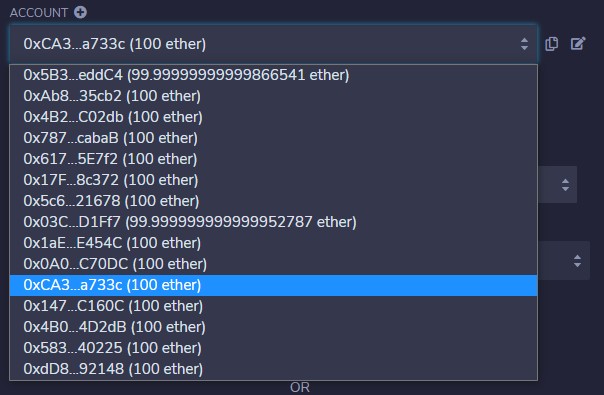
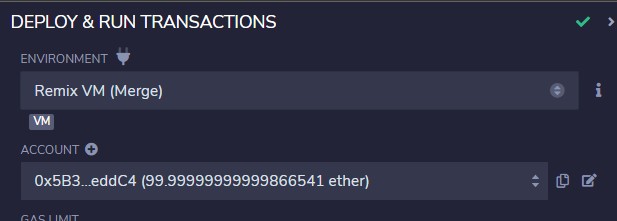
## **Step 1->** Click on owner to create an owner object



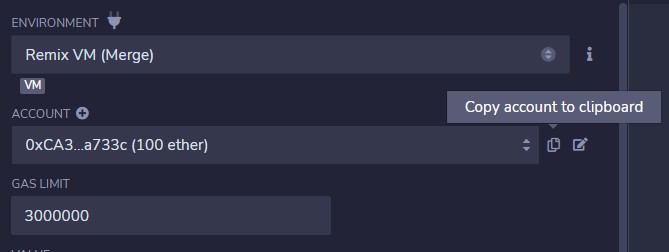
**Step 2->** Click on lastOwnerChange button



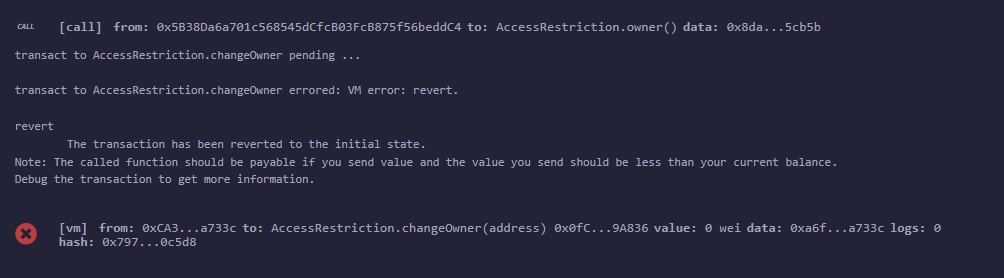
## **Step 3->** Change the address of the account from Account dropdown in Deploy tab of Remix IDE.

**Step 4->** Copy the address

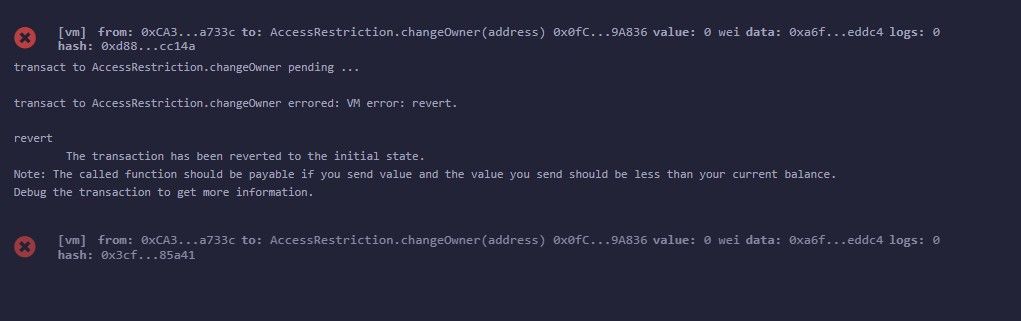


## **Step 5->** Paste the address in changeOwner input and click on changeOwner.

**Step 6->** You should get an error as following

## **Step 7->** If you click on buycontract it should give an error as follows

**Step 8->** Now, paste the actual address of the account in the changeowner input and click on changeowner



## **Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces**

### Contract

Output

pragma solidity ^0.5.0;

contract Contract\_demo {

string message = "Hello";

function dispMsg() public view returns (string memory) { return message;

}

}

### Inheritance

pragma solidity >=0.4.22 <0.6.0;

contract Parent {

uint256 internal sum;

function setValue() external { uint256 a = 10;

uint256 b = 20; sum = a + b;

}

}

contract child is Parent {

function getValue() external view returns (uint256) { return sum;

}

}

contract caller {

child cc = new child();

function testInheritance() public returns (uint256) { cc.setValue();

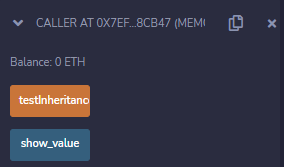
return cc.getValue();

}

function show\_value() public view returns (uint256) { return cc.getValue();

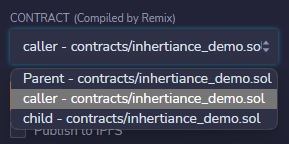
}

}

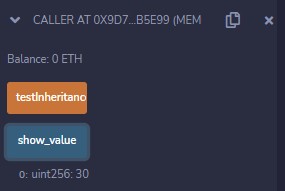
**Output:**

# Flow of execution

## **Step 1->** Select caller contract to deploy in Contract and deploy



**Step 2->** Click test Inheritance and then click on show\_value to view value



### 

### 3)Abstract Contracts

// SPDX-License-Identifier: MIT pragma solidity ^0.5.17;

contract Calculator {

function getResult() external view returns (uint256);

}

contract Test is Calculator { constructor() public {}

function getResult() external view returns (uint256) { uint256 a = 1;

uint256 b = 2;

uint256 result = a + b; return result;

}

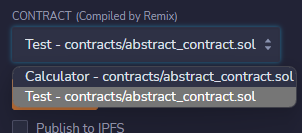
}

Outputs

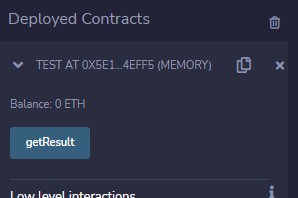
**Outputs:**

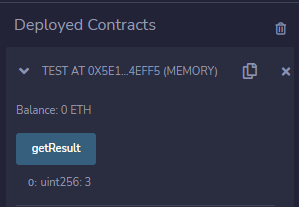
# Flow of execution

**Step 1->** Select Test contract and deploy



**Step 2->** The contact will deploy as below



**Step 3->** Click on getResult to get sum of a+b

### Constructors

// SPDX-License-Identifier: MIT pragma solidity ^0.5.0;

// Creating a contract

contract constructorExample { string str;

constructor() public {

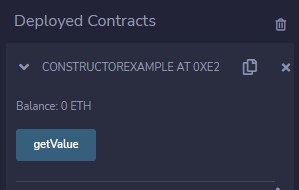
str = "GeeksForGeeks";

}

function getValue() public view returns (string memory) { return str;

}

}



# Flow of execution

**Step 1->** Click on getValue to print strin

### Interfaces

pragma solidity ^0.5.0;

interface Calculator {

function getResult() external view returns(uint);

}

contract Test is Calculator { constructor() public {}

function getResult() external view returns(uint){ uint a = 1;

uint b = 2;

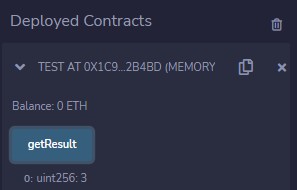
uint result = a + b; return result;

}

}

**Outputs:**

# Flow of execution

**Step 1->** Click on getResult to display sum

## **Libraries, Assembly, Events, Error handling.**

### Libraries myLib.sol Code

// SPDX-License-Identifier: MIT pragma solidity >=0.7.0 <0.9.0;

library myMathLib {

function sum(uint256 a, uint256 b) public pure returns (uint256) { return a + b;

}

function exponent(uint256 a, uint256 b) public pure returns (uint256) { return a\*\*b;

}

}

### using\_library.sol Code

// SPDX-License-Identifier: MIT pragma solidity >=0.7.0 <0.9.0;

import "contracts/myLIB.sol"; contract UseLib {

function getsum(uint256 x, uint256 y) public pure returns (uint256) {

return myMathLib.sum(x, y);

}

function getexponent(uint256 x, uint256 y) public pure returns (uint256) { return myMathLib.exponent(x, y);

}

}

**Outputs:**

# Flow of execution

## **Step 1->** Change contract to UseLib and deploy.

**Step 2->** The deployed contract should be same as below



## **Step 3->** Input values to both getexponent and getsum functions as below



**Step 4->** Execute both functions. You will get below outpu

### Assembly

// SPDX-License-Identifier: GPL-3.0 pragma solidity >=0.4.16 <0.9.0;

contract InlineAssembly {

// Defining function

function add(uint256 a) public view returns (uint256 b) { assembly {

let c := add(a, 16) mstore(0x80, c)

{

let d := add(sload(c), 12) b := d

}

b := add(b, c)

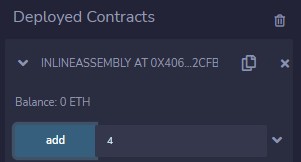
}

}

}



# Flow of execution

**Step 1->** Input a number for add function

**Step 2->** Click add to output sum



### Events

// SPDX-License-Identifier: MIT pragma solidity ^0.5.0;

// Creating a contract contract eventExample {

// Declaring state variables uint256 public value = 0;

// Declaring an event

event Increment(address owner);

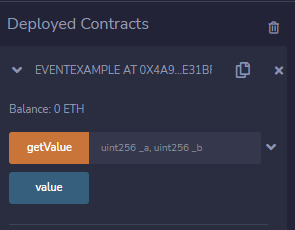
// Defining a function for logging event

function getValue(uint256 \_a, uint256 \_b) public { emit Increment(msg.sender);

value = \_a + \_b;

}

}

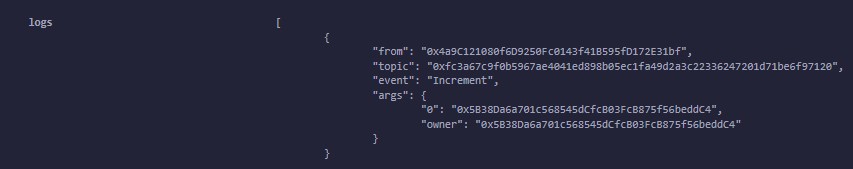
**Output**

# Flow of execution

## **Step 1->** Provide values to getValue function and click on it.



**Step 2->** In the terminal check for logs



### Error Handling

// SPDX-License-Identifier: MIT pragma solidity ^0.5.17;

contract ErrorDemo {

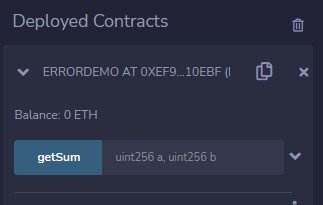
function getSum(uint256 a, uint256 b) public pure returns (uint256) { uint256 sum = a + b;

// require(sum < 255, "Invalid"); assert(sum<255);

return sum;

}

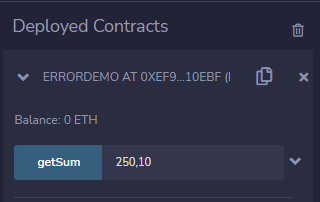
}



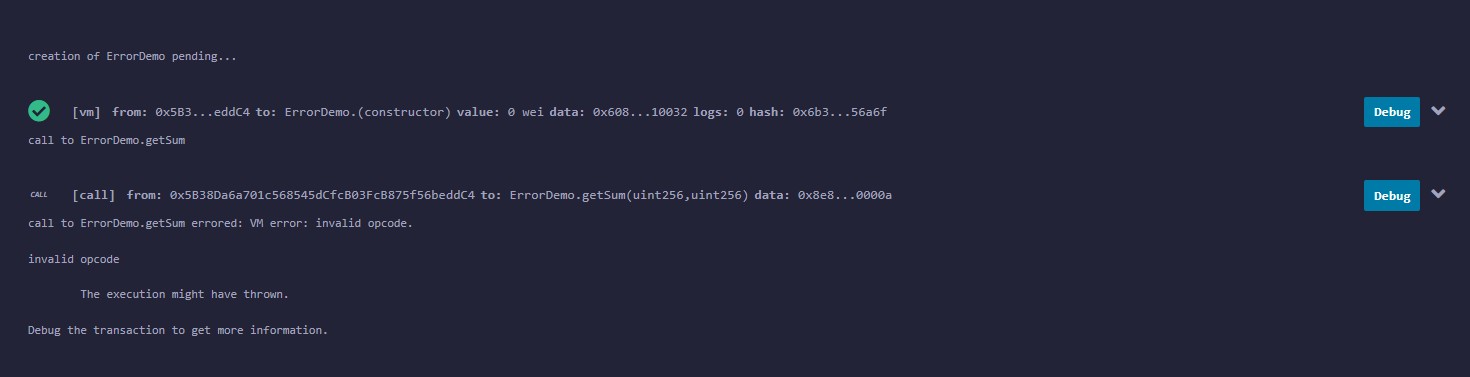
Output

# Flow of execution

## **Step 1->** Provide some values and press on getSum



**Step 2->** Check terminal panel



# PRACTICAL-5

# Aim: Write a program to demonstrate mining of ether

const Web3 = require('web3');

const web3 = new Web3(new

Web3.providers.HttpProvider('http: 127.0.0.1:7545')); Replace with your Ganache HTTP provider

async function mine() {

const accounts = await web3.eth.getAccounts(); const coinbaseacc1 = accounts[0];

const coinbaseacc2 = accounts[1];

console.log(`Mining ether on Ganache with coinbase address:

${coinbaseacc1}`);

while (true) { try {

await web3.eth.sendTransaction({ from: coinbaseacc1,

to: coinbaseacc2, value: 50,

});

console.log(`Mined a new block!`);

} catch (err) { console.error(err);

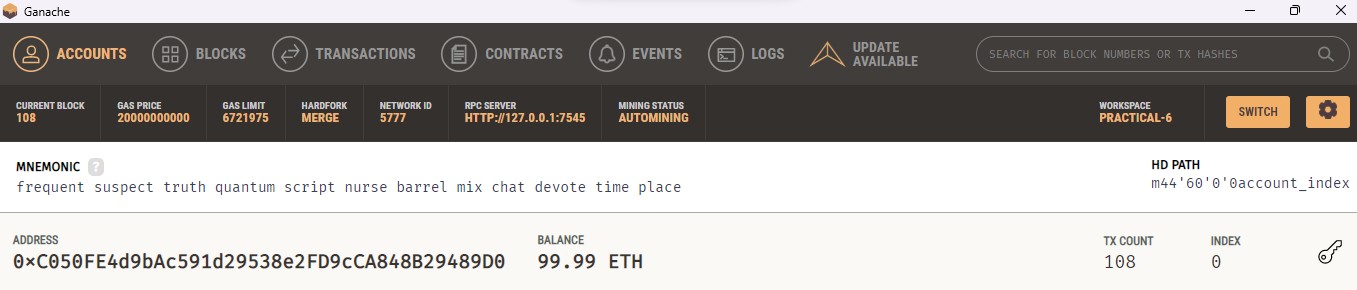
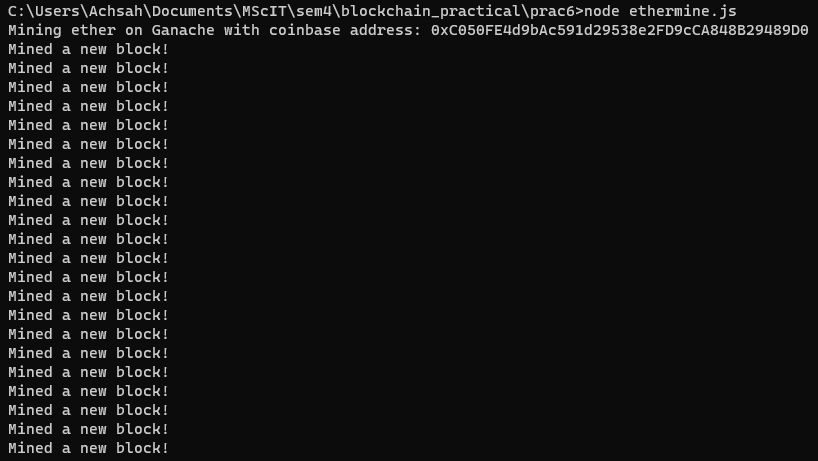
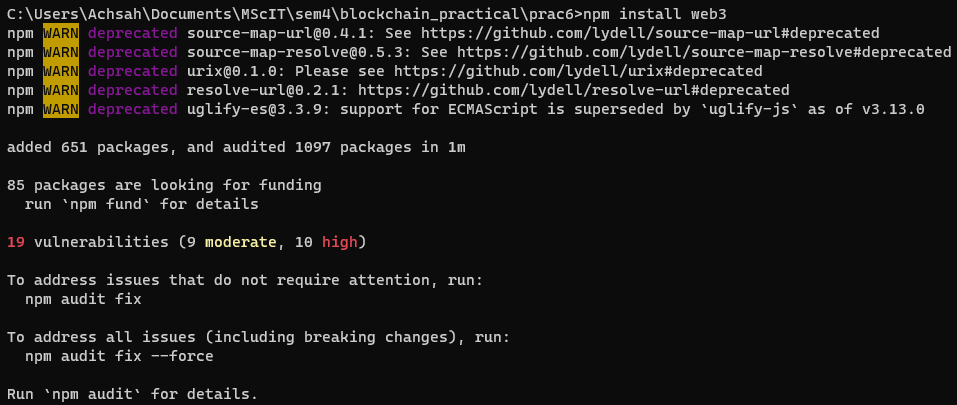
}

}

}

mine();

**Output:**



# PRACTICAL-6

# Aim: Demonstrate the running of the blockchain node

## **Step 1->**Create a folder named ethermine and a JSON file named genesis.json and write the following lines in it.

{

"config": {

"chainId": 3792,

"homesteadBlock": 0,

"eip150Block": 0,

"eip155Block": 0,

"eip158Block": 0

},

"difficulty": "2000",

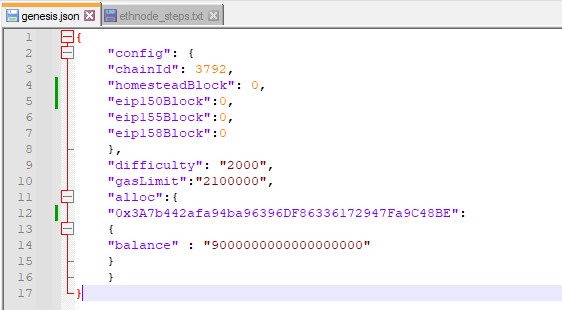
"gasLimit": "2100000", "alloc": {

"0×0b6C4c81f58B8d692A7B46AD1e16a1147c25299F": { "balance": "9000000000000000000"

}

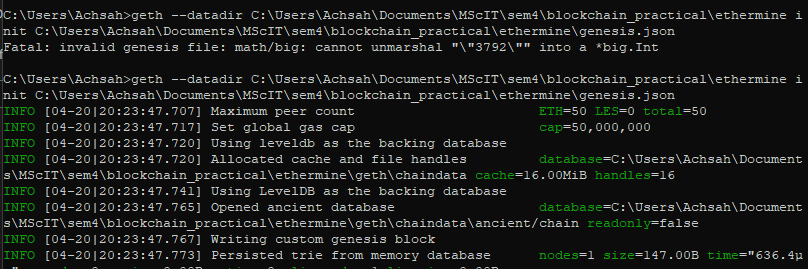
}

}



# Step 2->Run command geth account new –datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain\_practical\ethermine testnet-blockchain

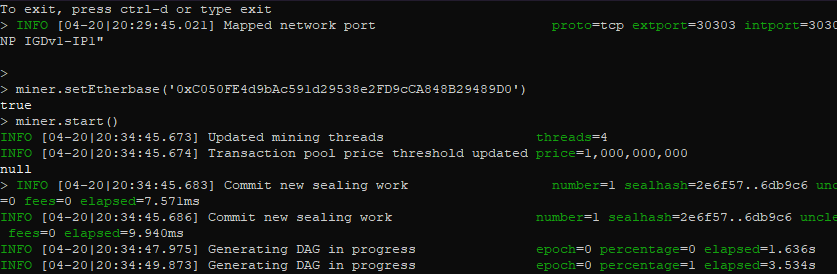
**Step 3->** Run command **geth account new --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain\_practical\ethermine**



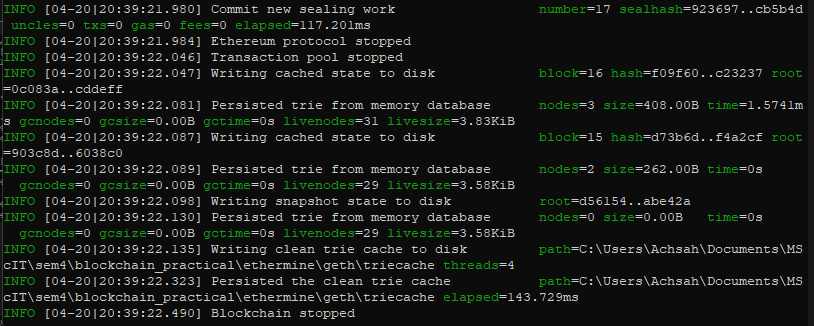
# Step 4->Run command geth --identity "localB" --http --http.port "8280"--http.corsdomain "\*" --http.api"db,eth,net,web3" –datadir "C:\Users\Achsah\Documents\MScIT\sem4\blockchain\_practical\ethermine" --port "30303" - nodiscover --networkid 5777 console. This command will enable geth console.

**Step 5->** Run the command **miner.setEtherbase('0xC050FE4d9bAc591d29538e2FD9cCA848B29489D0’)** in the geth console

**Step 6->** Run the command **miner.start()** to start mining



## **Step 7->** Below screenshots are the mining processes running on your local machine.

**Step 8->** To stop the mining press **Ctrl+D**

# PRACTICAL-7

# Aim: Create your own blockchain and demonstrate its use

### Create a javascript folder with the following code in any folder of your choice.

# JavaScript Code

const SHA256 = require("crypto-js/sha256"); class Block {

constructor(index, timestamp, data, previousHash = "") { this.index = index;

this.timestamp = timestamp; this.data = data; this.previousHash = previousHash; this.hash = this.calculateHash();

}

calculateHash() { return SHA256(

this.index + this.previousHash + this.timestamp + JSON.stringify(this.data)

).toString();

}

}

class Blockchain { constructor() {

this.chain = [this.createGenesisBlock()];

}

createGenesisBlock() {

return new Block(0, "21/04/2023", "Genesis Block", "0");

}

getLatestBlock() {

return this.chain[this.chain.length - 1];

}

addBlock(newBlock) {

newBlock.previousHash = this.getLatestBlock().hash;

newBlock.hash = newBlock.calculateHash(); this.chain.push(newBlock);

}

isChainValid() {

for (let i = 1; i < this.chain.length; i +) { const currentBlock = this.chain[i];

const previousBlock = this.chain[i - 1];

if (currentBlock.hash currentBlock.calculateHash()) { return false;

}

if (currentBlock.previousHash previousBlock.hash) { return false;

}

}

return true;

}

}

Blockchain Implementation

let myCoin = new Blockchain();

myCoin.addBlock(new Block(1, "22/04/2023", { amount: 4 })); myCoin.addBlock(new Block(2, "22/04/2023", { amount: 8 }));

console.log('Is blockchain valid? ' + myCoin.isChainValid()); console.log(JSON.stringify(myCoin, null, 4));

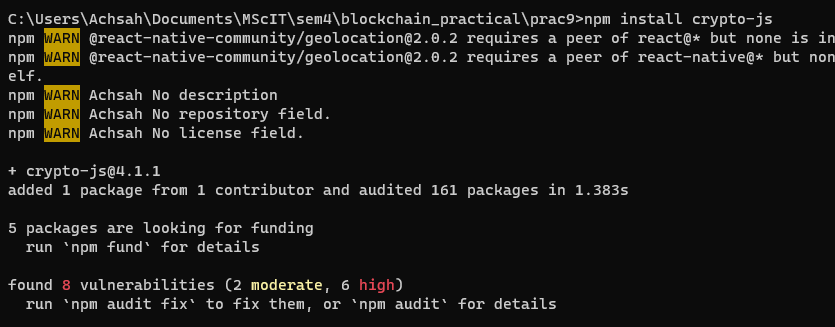
Output

# Flow of execution

## **Step 1->** Make sure you have installed nodejs in your system



**Step 2->** We need **crypto –js** node module to make our own blockchain. So install it as following



## **Step 3->** Run the above code in command line using command: node main.js

