INDEX

| **List of Practical:** | | **Date** | **Signature** |
| --- | --- | --- | --- |
| 1. | Write the following programs for Blockchain in Python: |  |  |
| a. | A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it. | 9/2/23 |  |
| b. | A transaction class to send and receive money and test it. |
| c. | Create multiple transactions and display them. |
| d. | Create a blockchain, a genesis block and execute it. |
| e. | Create a mining function and test it. |
| f. | Add blocks to the miner and dump the blockchain. |
| 2. | Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.(MetaMask & Remix | 24/2/23 |  |
| 3. | Implement and demonstrate the use of the following in Solidity: |  |  |
| a. | Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs,Mappings, Conversions, Ether Units, Special Variables. | 10/3/23 |  |
| b. | Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions. |
| 4. | Implement and demonstrate the use of the following in Solidity: |  |  |
| a. | Withdrawal Pattern, Restricted Access. | 20/3/23 |  |
| b. | Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces. |
| c. | Libraries, Assembly, Events, Error handling. |
| 5. | Write a program to demonstrate mining of Ether. | 27/3/23 |  |
| 6. | Demonstrate the running of the blockchain node. | 03/04/23 |  |
| 7. | Create your own blockchain and demonstrate its use. | 10/04/23 |  |

**Table of Contents**

Practical -1 Write the following programs for Blockchain in Python 5

1. [A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it 5](#_30j0zll)
2. [A transaction class to send and receive money and test it 6](#_1fob9te)
3. [Create multiple transactions and display them 8](#_3znysh7)
4. [Create a blockchain, a genesis block and execute it 10](#_2et92p0)
5. [Create a mining function and test it 11](#_tyjcwt)
6. [Add blocks to the miner and dump the blockchain 12](#_3dy6vkm)

Practical-2 Install and configure Go Ethereum and the Mist browser. Develop and test a sample application(MetaMask & Remix) 14

Practical-3 Implement and demonstrate the use of the following in Solidity 26

1. [Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables 26](#_1t3h5sf)
   1. [Variable 26](#_4d34og8)
   2. [Strings 27](#_2s8eyo1)
   3. [Operators 28](#_17dp8vu)
   4. [Array 29](#_3rdcrjn)
   5. [Decision Making 30](#_26in1rg)
   6. [Loops 31](#_lnxbz9)
   7. [Enums 34](#_35nkun2)
   8. [Structs 35](#_1ksv4uv)
   9. [Mappings 36](#_44sinio)
   10. [Conversions 37](#_2jxsxqh)
   11. [Ether Units 38](#_z337ya)
   12. [Special Variables 41](#_3j2qqm3)
2. Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions 42
   1. [View Functions 42](#_1y810tw)
   2. [Pure Functions 43](#_4i7ojhp)
   3. [Mathematical Functions 44](#_2xcytpi)
   4. [Cryptographic Functions 45](#_1ci93xb)
   5. [Functions 46](#_3whwml4)
   6. [Fallback Function 48](#_2bn6wsx)
   7. [Function Overloading 50](#_qsh70q)
   8. [Function modifiers 51](#_3as4poj)

Practical-4 Implement and demonstrate the use of the following in Solidity 53

1. [Withdrawal Pattern 53](#_1pxezwc)

[Flow of execution 54](#_49x2ik5)

1. [Restricted Access 57](#_2p2csry)

[Flow of execution 58](#_147n2zr)

1. [Contracts 62](#_3o7alnk)
2. [Inheritance 62](#_23ckvvd)

[Flow of execution 63](#_ihv636)

1. [Abstract Contracts 64](#_32hioqz)

[Flow of execution 64](#_1hmsyys)

1. [Constructors 65](#_41mghml)

[Flow of execution 66](#_2grqrue)

1. [Interfaces 66](#_vx1227)

[Flow of execution 67](#_3fwokq0)

1. Libraries 68

myLib.sol Code 68

[using\_library.sol Code 68](#_1v1yuxt)

[Flow of execution 69](#_4f1mdlm)

1. [Assembly 70](#_2u6wntf)

[Flow of execution 71](#_19c6y18)

1. [Events 72](#_3tbugp1)

[Flow of execution 73](#_28h4qwu)

1. [Error Handling 73](#_nmf14n)

[Flow of execution 74](#_37m2jsg)

[Practical-5 write a program to demonstrate mining of ether 75](#_1mrcu09)

Practical-6 Demonstrate the running of the blockchain node 77

Practical-7 Create your own blockchain and demonstrate its use 81

[Create a javascript folder with the following code in any folder of your choice. 81](#_46r0co2)

[JavaScript Code 81](#_2lwamvv)

[Flow of execution 83](#_111kx3o)

# PRACTICAL -1 WRITE THE FOLLOWING PROGRAMS FOR BLOCKCHAIN

**IN PYTHON**

### 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

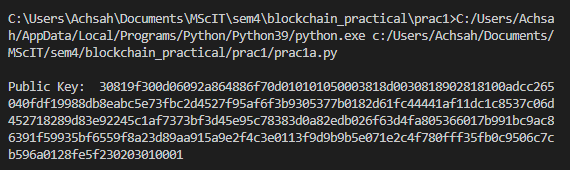
binascii.hexlify(self.\_public\_key.exportKey(format="DER")).decode( "ascii"

)

Dinesh = Client()

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





### 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)



### 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]

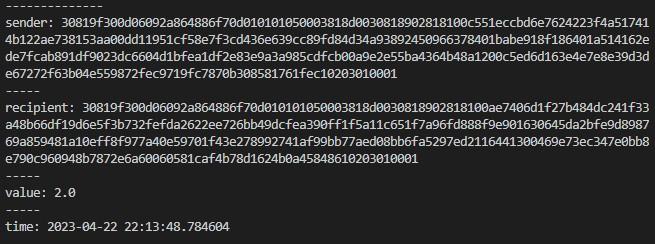
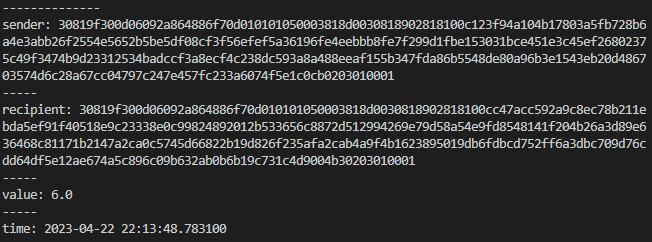
t2 = Transaction(Dinesh, Seema.identity, 6.0) t2.sign\_transaction() transactions.append(t2)

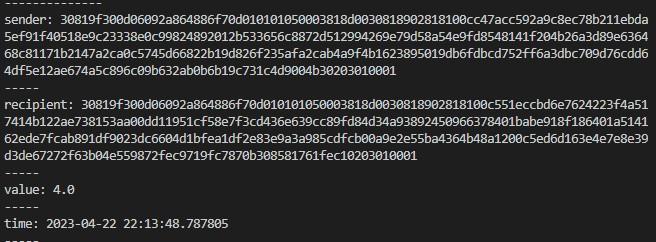
t3 = Transaction(Ramesh, Vijay.identity, 2.0) t3.sign\_transaction() transactions.append(t3)

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

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







### 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)

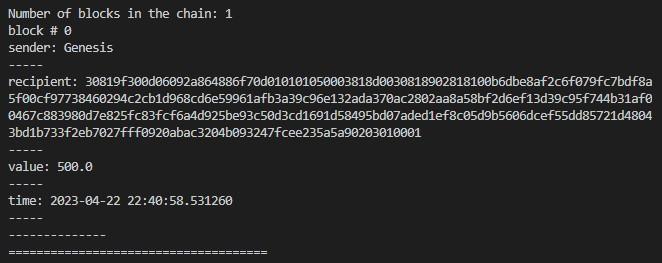
block0 = Block(Dinesh) block0.previous\_block\_hash = ""

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)



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))

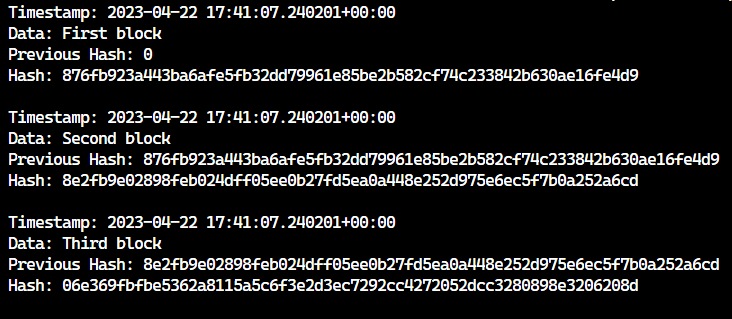
# Dumping the blockchain

for block in blockchain: print(

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

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

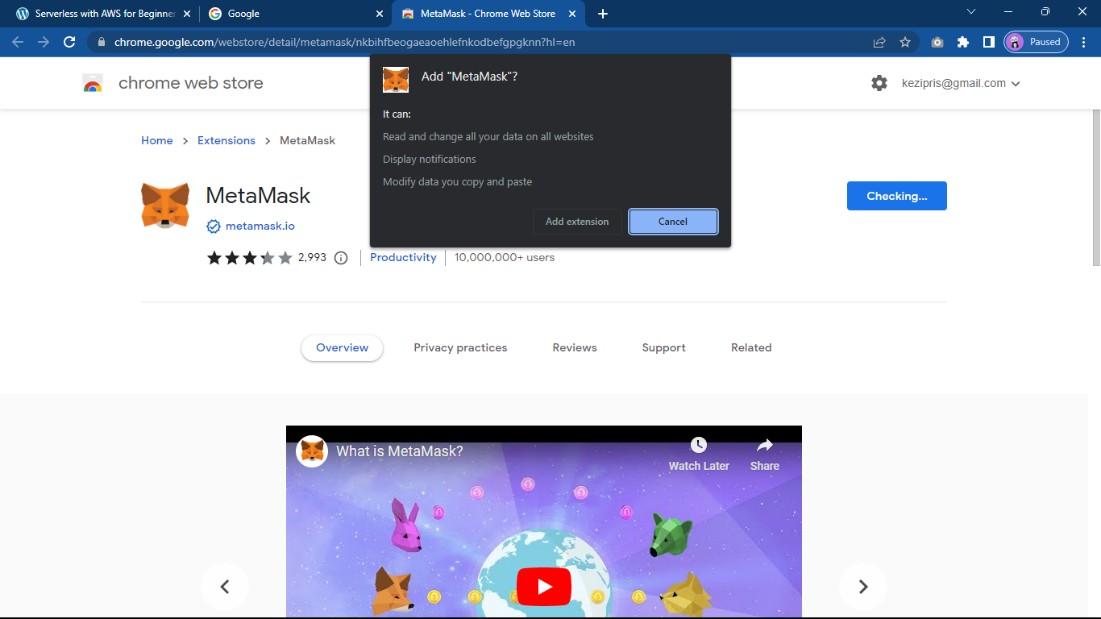
)

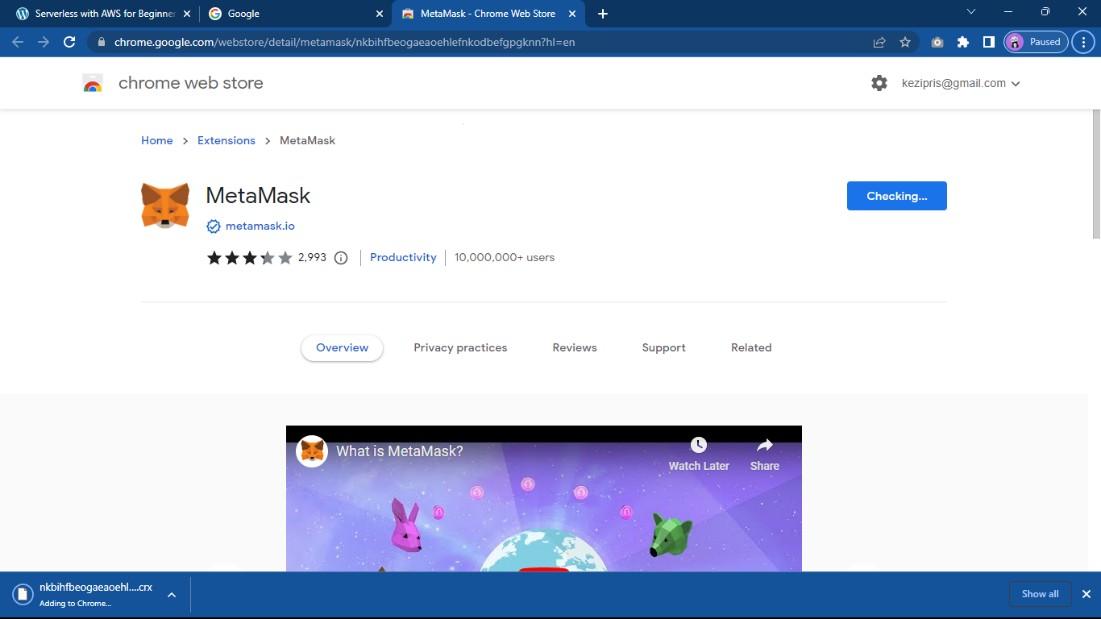


# PRACTICAL-2 INSTALL AND CONFIGURE GO ETHEREUM AND THE MIST BROWSER. DEVELOP AND TEST A SAMPLE

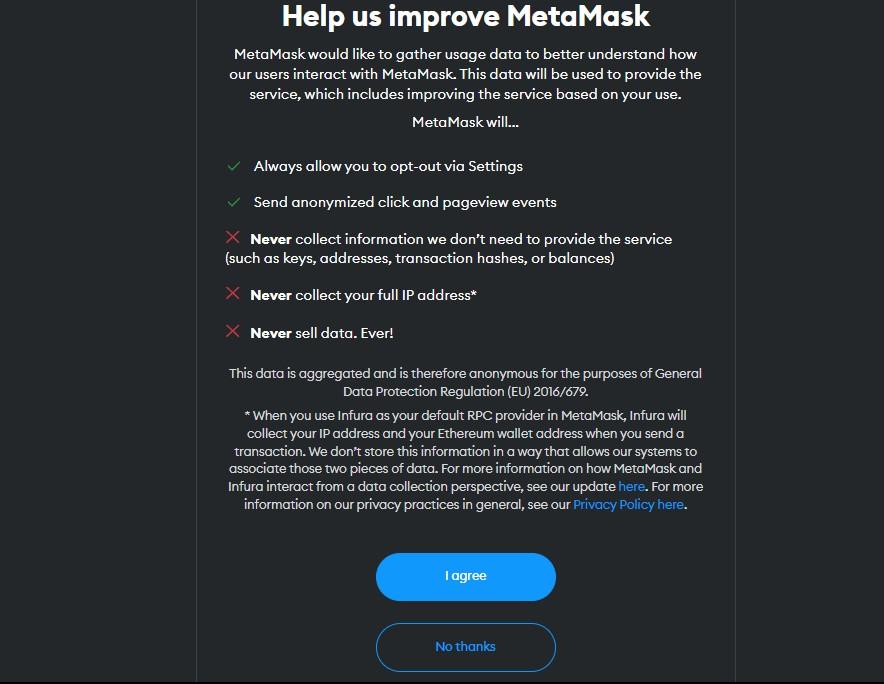
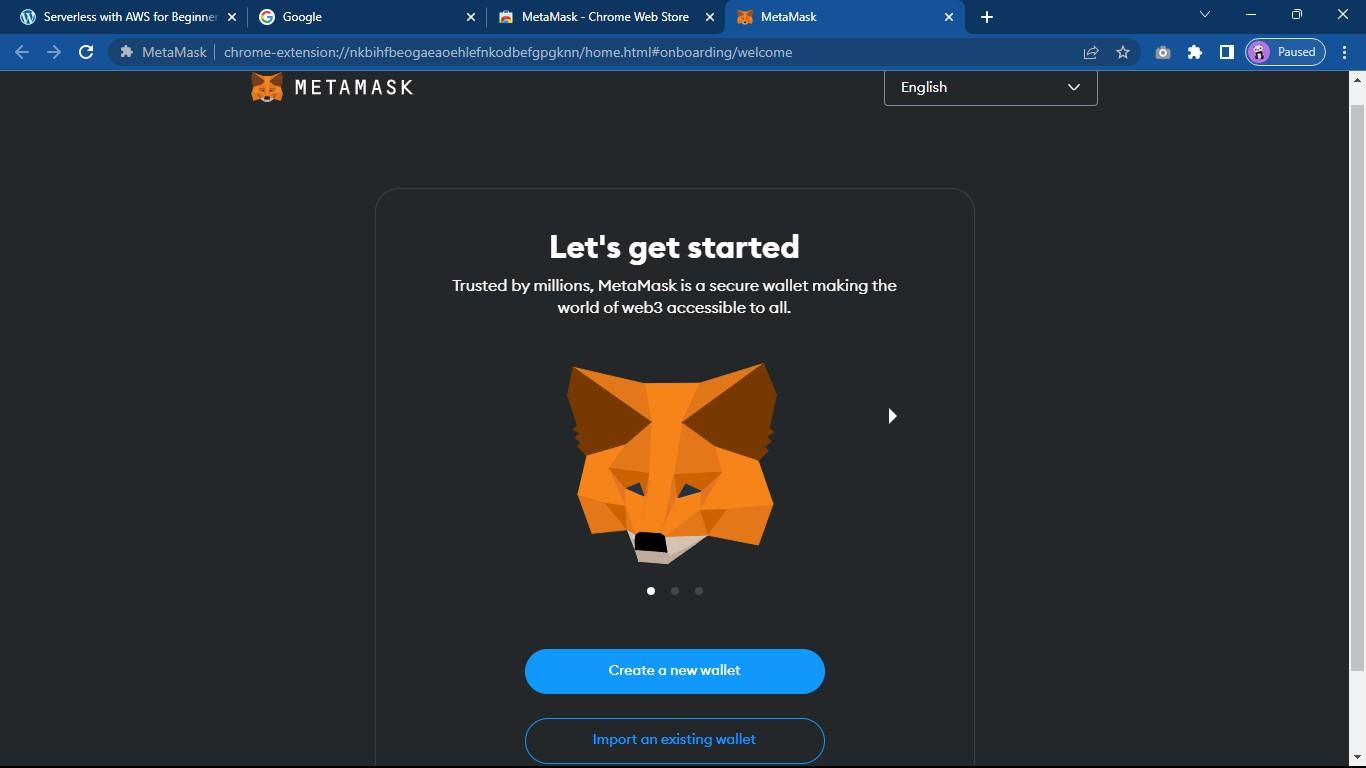
**APPLICATION(METAMASK & REMIX)**

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

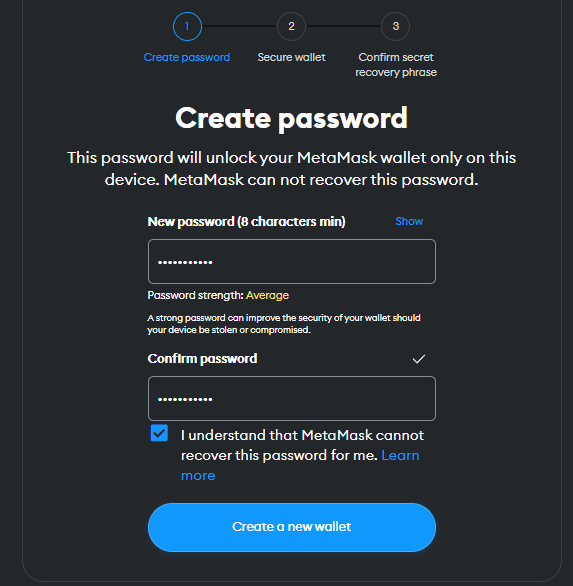


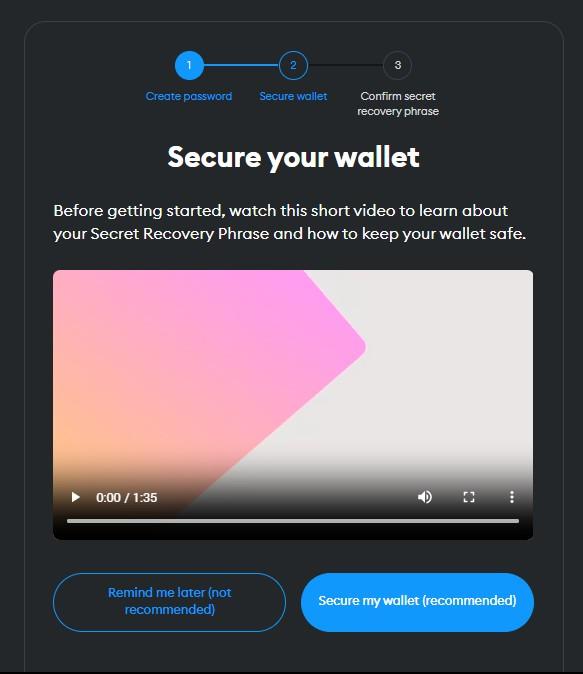


**Step 2->** Click on Metamask Extension in Extensions. Below page will open in a new tab. Click on Create a New Wallet. Click on I agree.

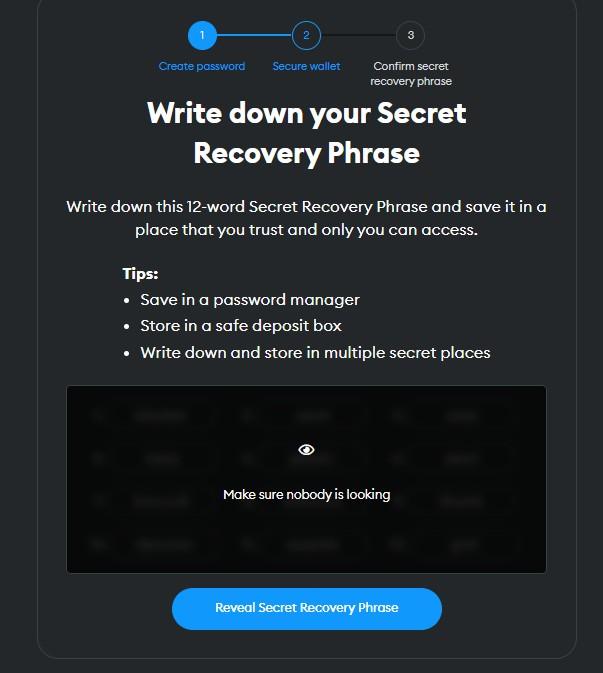


## **Step 3->** Create a password. This password can be used only on the device it was created on. Create a Strong password and click on Create a new Wallet button

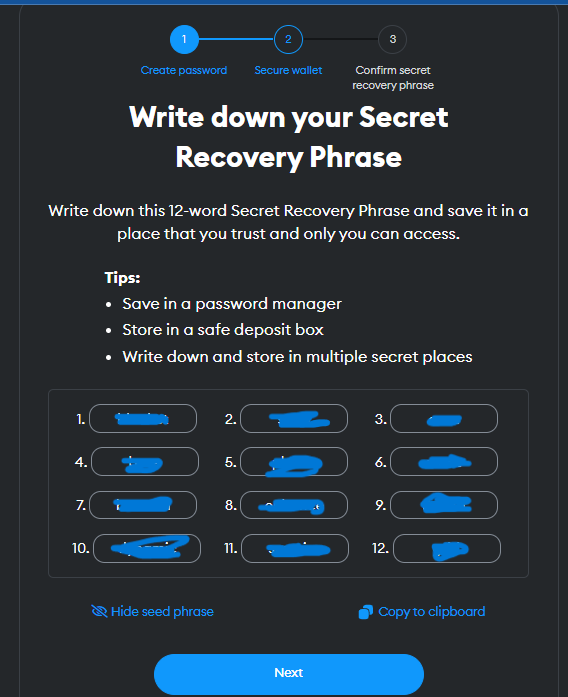




**Step 4->** Click on Secure my wallet button, following window will appear



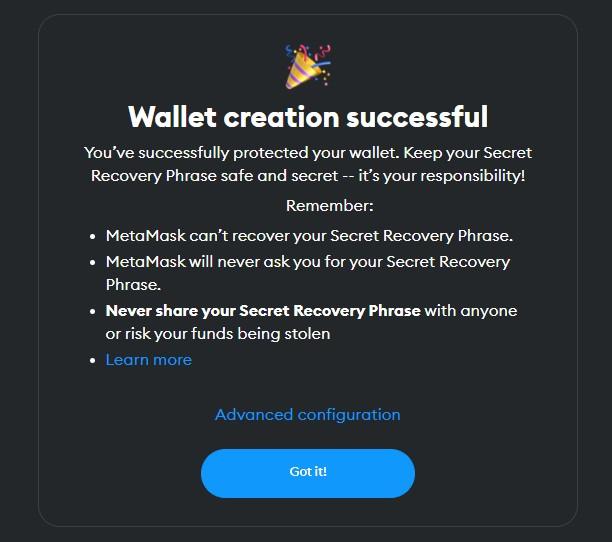
## **Step 5->** Click on Reveal Secret Recovery Phrase button and save the words in the same sequence



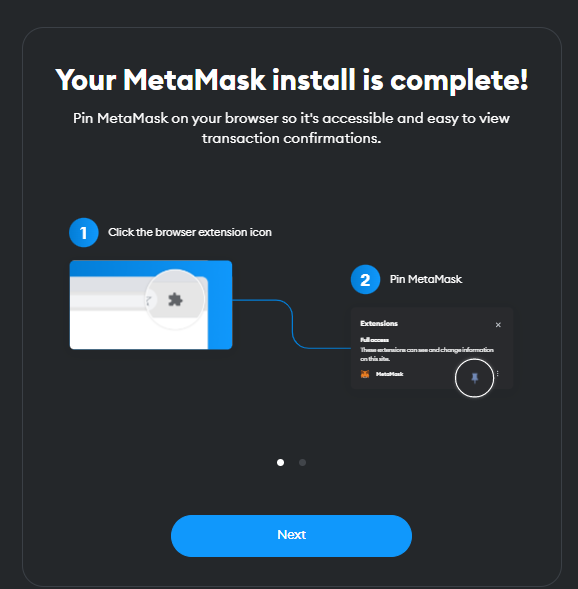
**Step 6->** Enter the respective words in the empty positions and click Confirm.



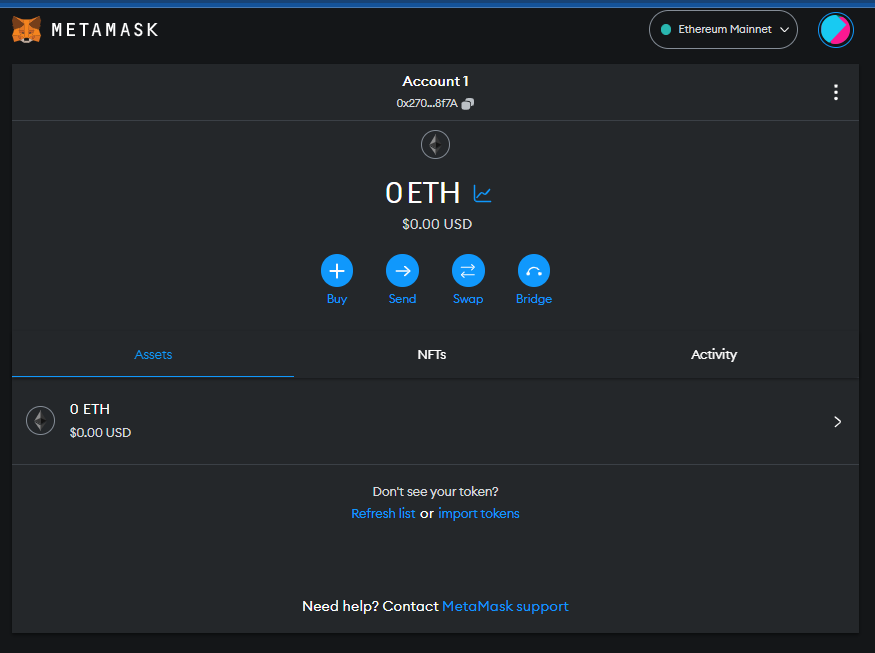
**Step 7->** Click Got it!



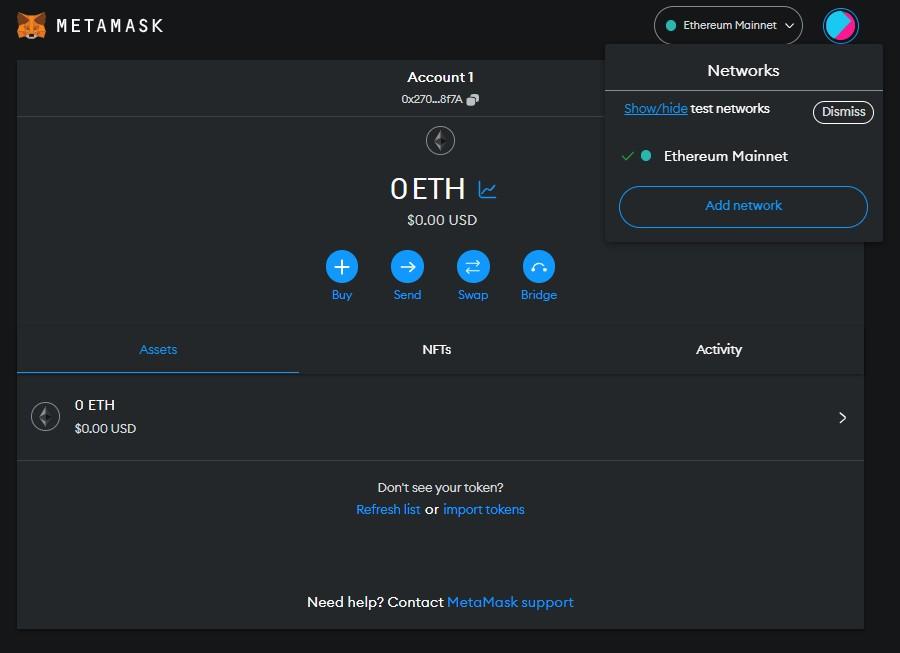
**Step 8->** Click on Next

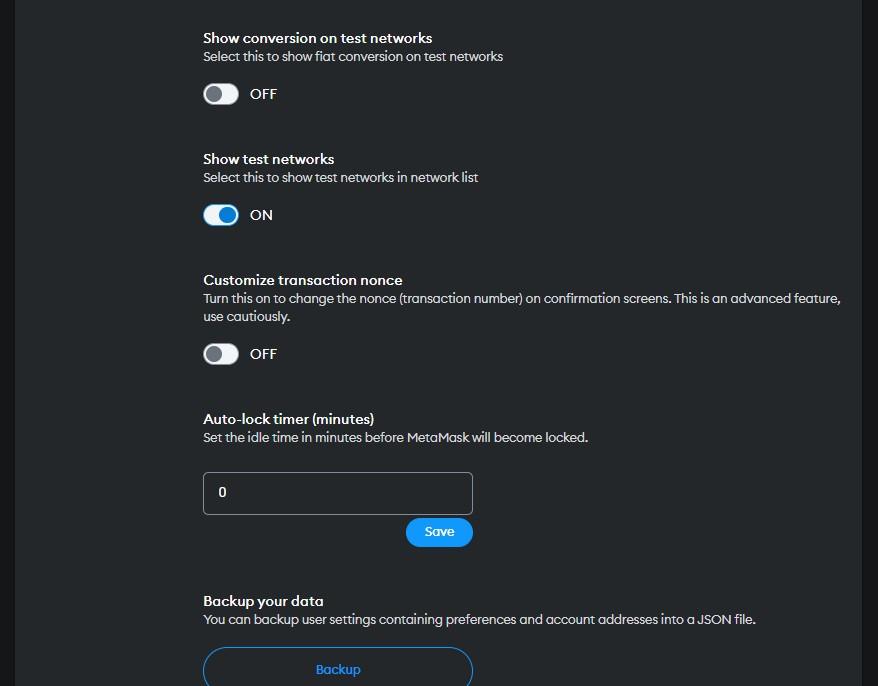


**Step 9->** Following will be the Dashboard

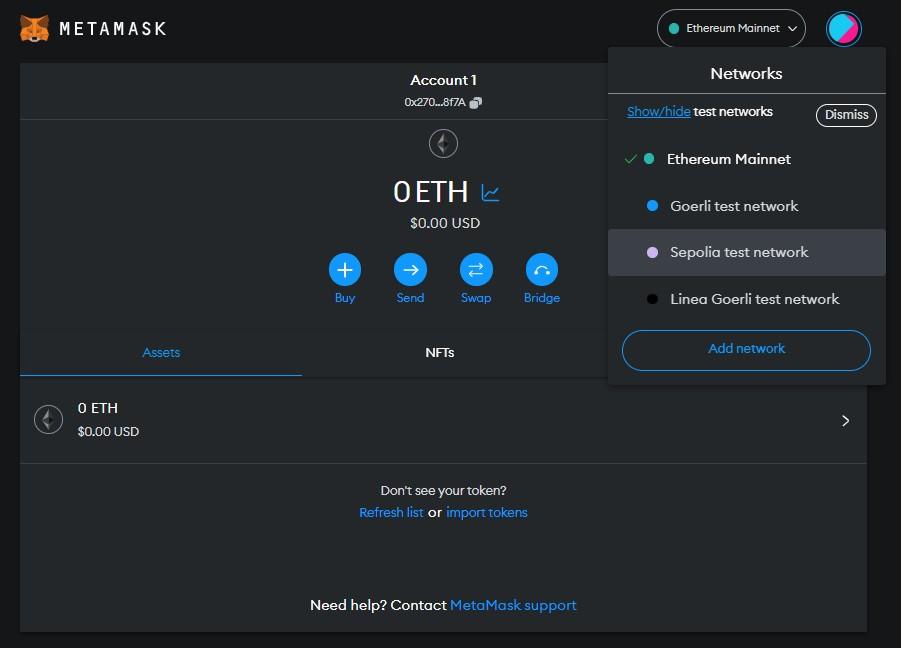


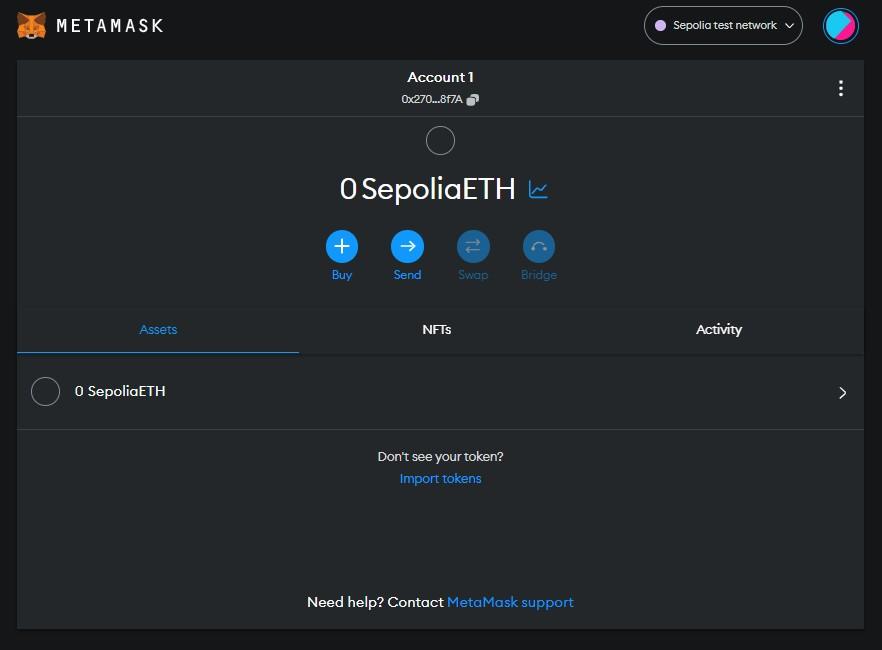
## **Step 10->** Click on Ethereum Mainnet button. Next click on Show/hide test networks.



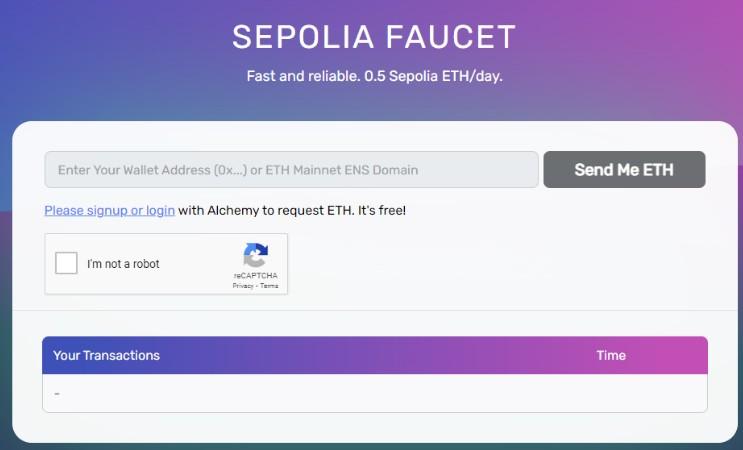


**Step 11->** Check if tesnets are shown by clicking on Etherum Mainnet button. Click on Sepolia test network.

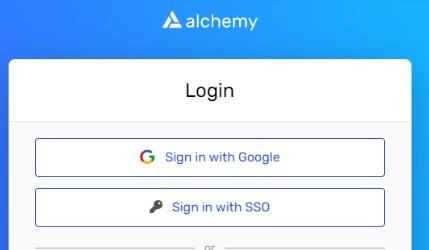




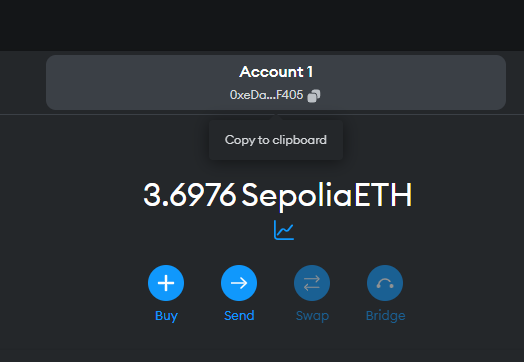
**Step 12->** Go to <https://sepoliafaucet.com/>and Click on Alchemy Login button.



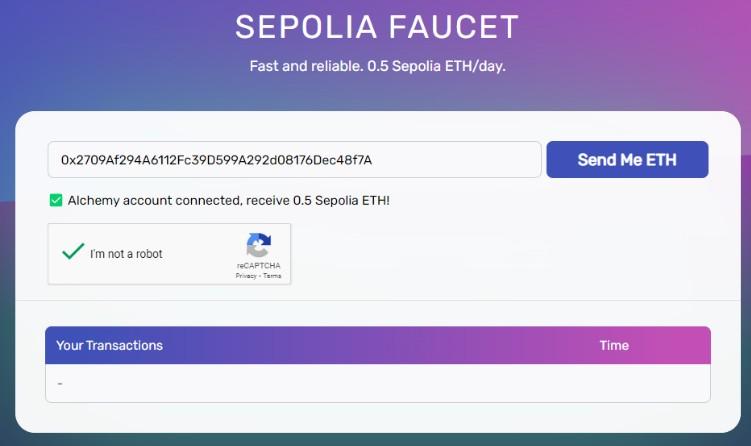
**Step 13->** Login to a gmail account in another browser tab and click on Sign in with Google



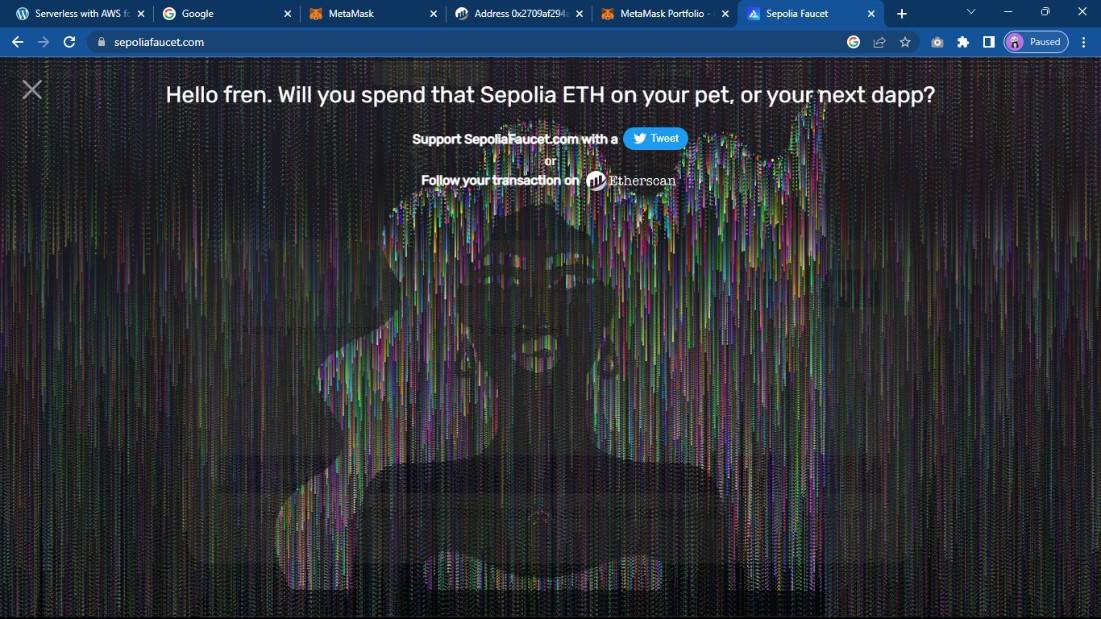
## **Step 14->** Now go to MetaMask and copy the account address.



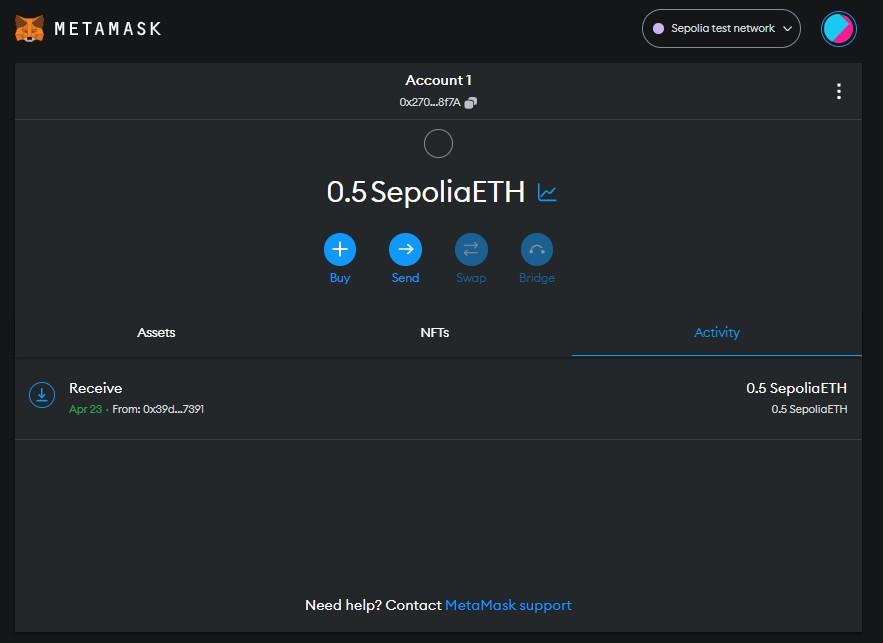
**Step 15->** Paste the address and click on Send Me ETH.



## **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.



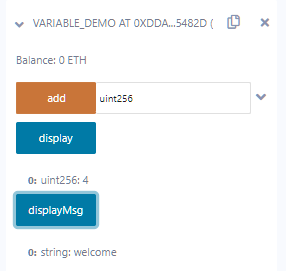
# PRACTICAL-3 IMPLEMENT AND DEMONSTRATE THE USE OF THE

**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**

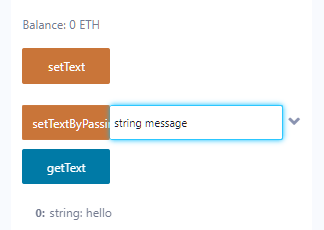
### Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables

# Variable

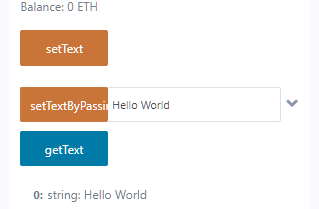


**FIGURE 1 -DISPLAYING VARIABLE VALUE**

# Strings

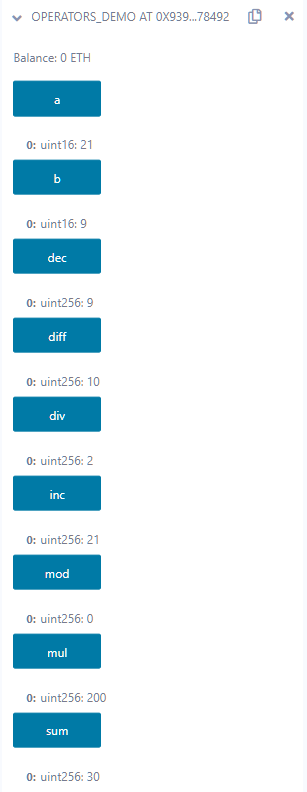


**FIGURE 2 - BEFORE SETTING NEW STRING VALUE**



**FIGURE 3 - AFTER SETTING STRING VALUE**

# Operators



**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;

}

}

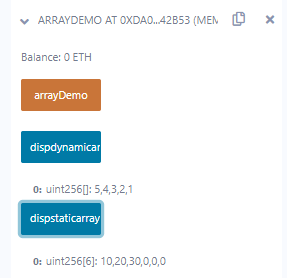
function dispdynamicarray() public view returns(uint[] memory)

{

return arr1;

}

}



**FIGURE 5 - ARRAY DISPLAYED**

# Decision Making

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;

}

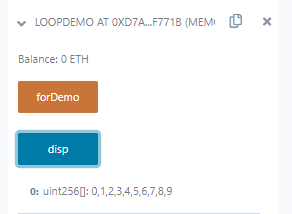
function disp() public view returns(uint[] memory)

{

return data;

}

}



**FIGURE 7 - APPENDING VALUES TO ARRAY USING FOR LOOP**

pragma solidity ^0.5.0; contract whiledemo

{

uint [] data; uint x=0;

function whileLoopDemo() public

{

while(x<5)

{

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

}

}

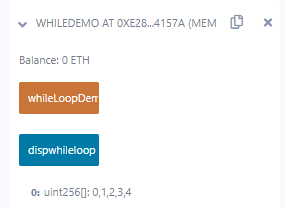
function dispwhileloop() public view returns(uint[] memory)

{

return data;

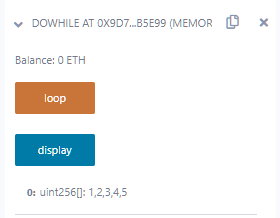
}

}



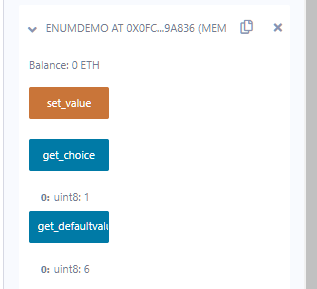
**FIGURE 8 - APPENDING VALUES TO ARRAY USING WHILE LOOP**

## Do



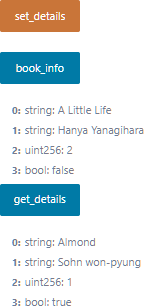
**FIGURE 9 APPENDING VALUES TO ARRAY USING DO WHILE LOOP**

# Enums



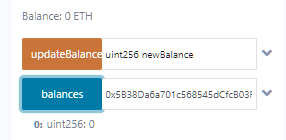
**FIGURE 10 - ACCESSING ENUM VALUES**

# Structs

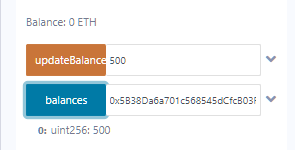


**FIGURE 11- STRUCTURE DATATYPE IN SOLIDITY**

# Mappings



**FIGURE 12 - BEFORE UPDATING BALANCE**



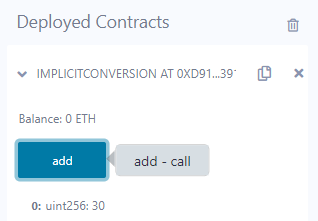
**FIGURE 13 - AFTER UPDATING BALANCE**

# Conversions

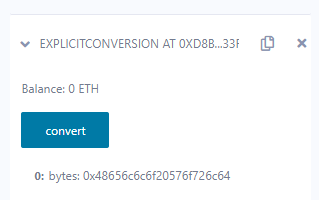
**Step 1->** Deploy both contracts



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

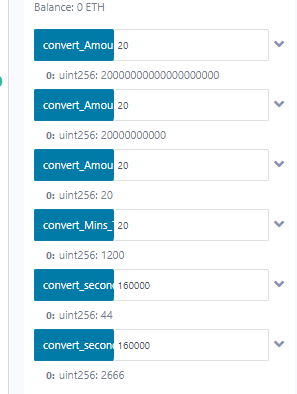


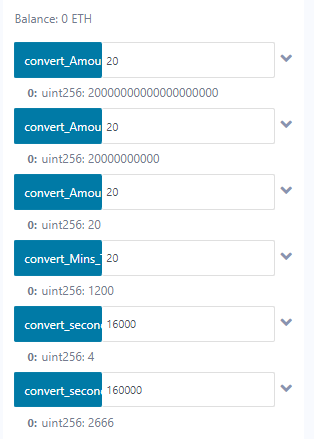
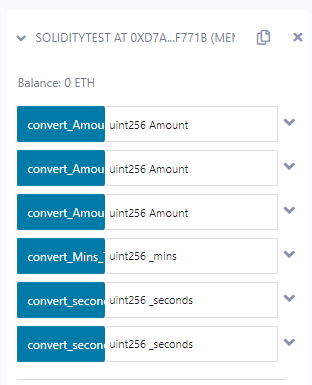
**Step 3->** Open Explicit Conversion and click on convert button



# Ether Units



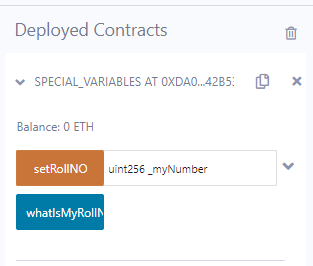


**Step 1->** Provide values to each function and click on them

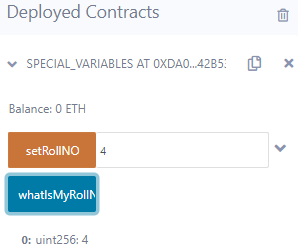
# Special Variables



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

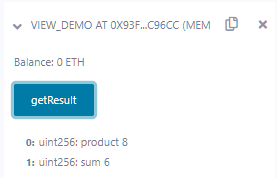


## **Step 2->** Input a number for setRollNO function and click on it & whatIsMyRollNumber button



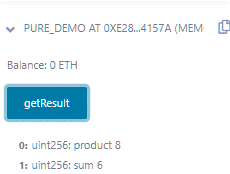
1. Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions

# View Functions



**FIGURE 14 - VIEW FUNCTION DEMO**

# Pure Functions



**FIGURE 15 - PURE FUNCTION OUTPUT**

# Mathematical Functions

pragma solidity ^0.5.0; contract Test{

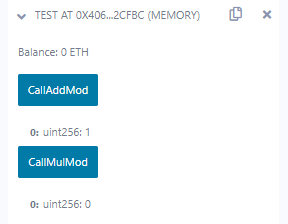
function CallAddMod() public pure returns(uint){ return addmod(7,3,3);

}

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");

}

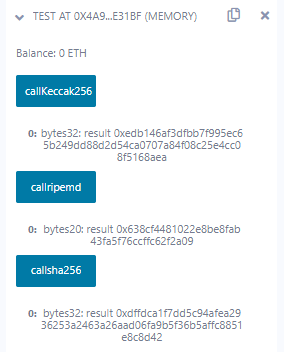
function callsha256() public pure returns(bytes32 result){ return sha256("BLOCKCHAIN");

}

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

}

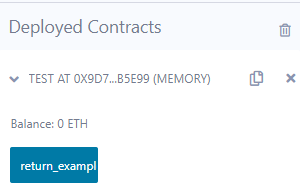
}



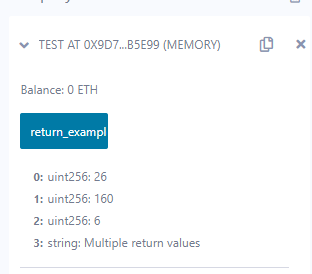
**FIGURE 17 - CRYPTOGRAPHY ALGORITHMS IN SOLIDITY**

# Functions

**Step 1->** Deploy Test Contract



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

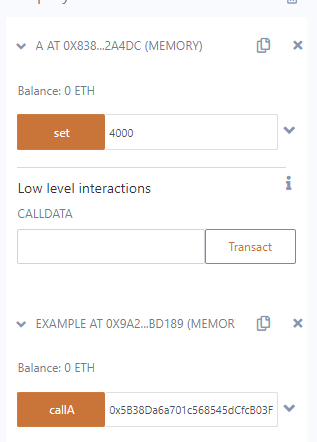


# Fallback Function

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

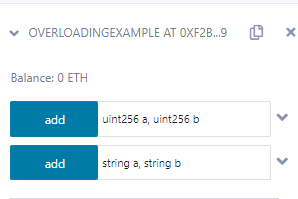


**Step 2->** Provide values to both deployed contracts accordingly(use any address)

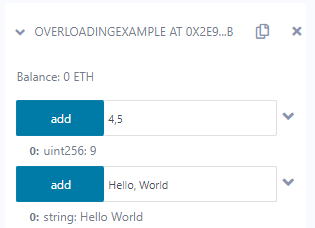


# Function Overloading

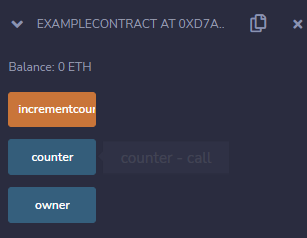
**Step 1->** Deploy Overloading Example contract



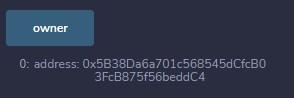
**Step 2->** Give integer and string values to both add functions as below



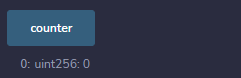
# Function modifiers



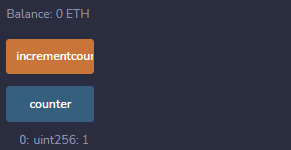
**Step 1->** Click on owner button



## **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 IMPLEMENT AND DEMONSTRATE THE USE OF THE

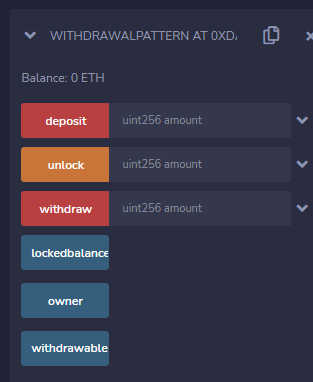
**FOLLOWING IN SOLIDITY**

## Withdrawal Pattern, Restricted Access

### Withdrawal Pattern

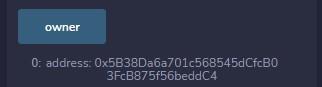




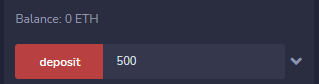


# Flow of execution

**Step 1->** Click on owner



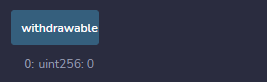
## **Step 2->** Enter an amount and click on deposit



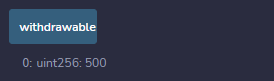
**Step 3->** Click on locked balance button to display the locked amount in the account



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

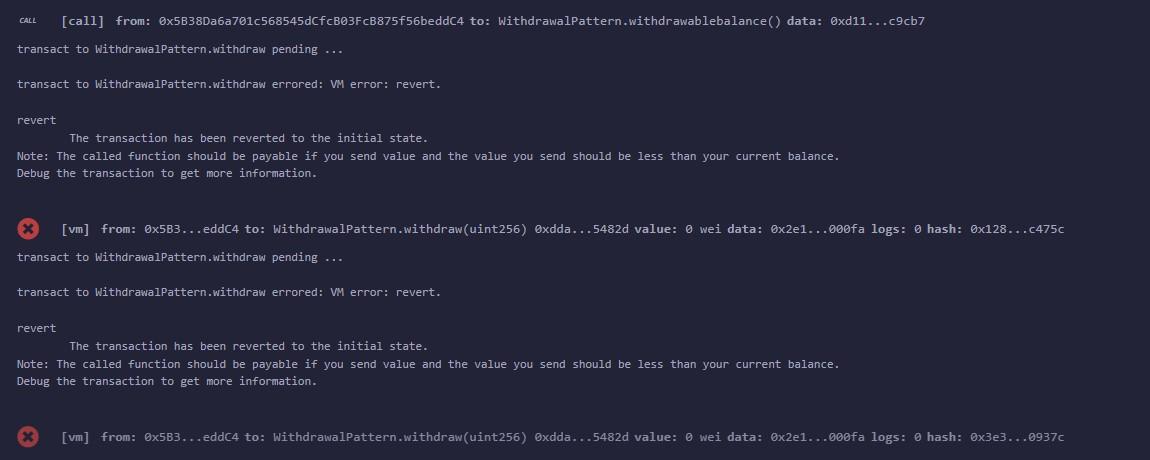


**Step 5->** Click on unlock button and enter any amount to transfer amount to withdrawable balance. Check locked balance and withdrawable balance.



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

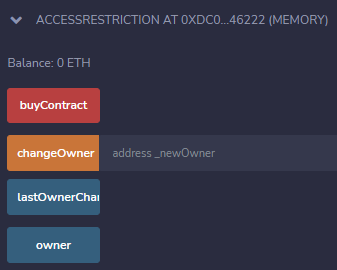
You should get an error and the transaction should be reverted.



### Restricted Access





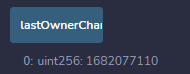


# Flow of execution

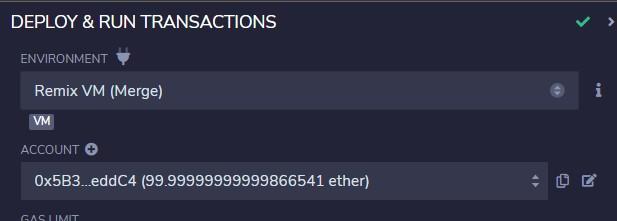
## **Step 1->** Click on owner to create an owner object

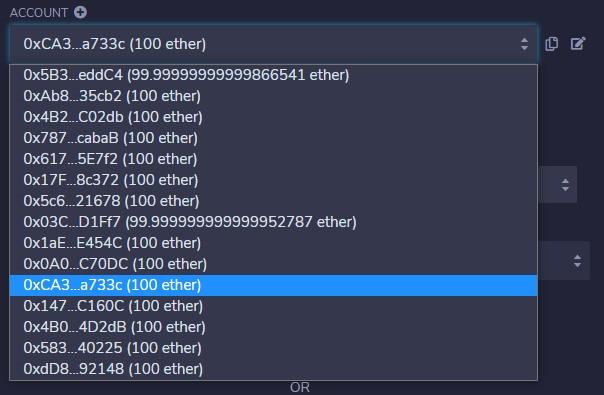


**Step 2->** Click on lastOwnerChange button

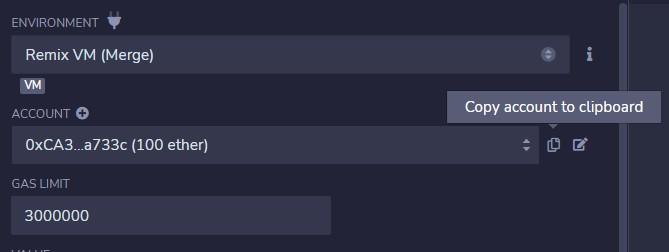


## **Step 3->** Change the address of the account from Account dropdown in Deploy tab of Remix IDE.

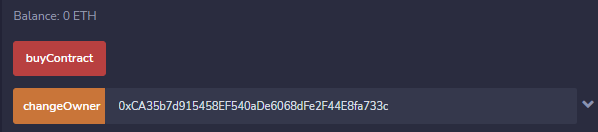




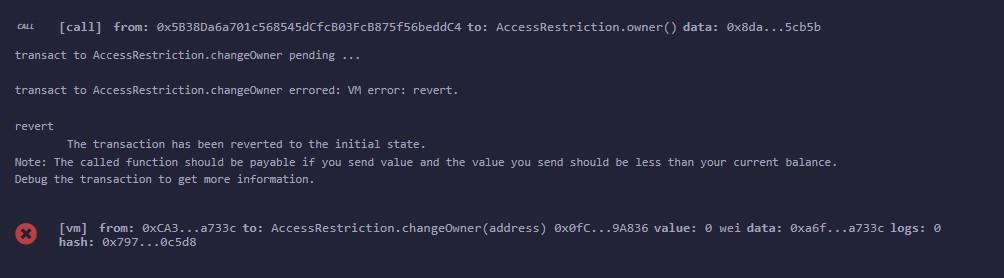
**Step 4->** Copy the address



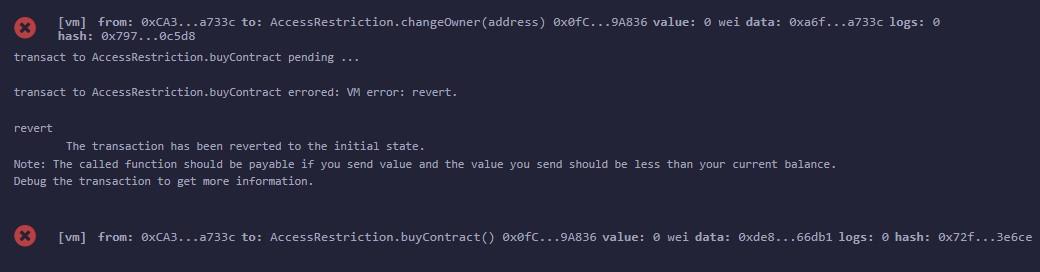
## **Step 5->** Paste the address in changeOwner input and click on changeOwner.



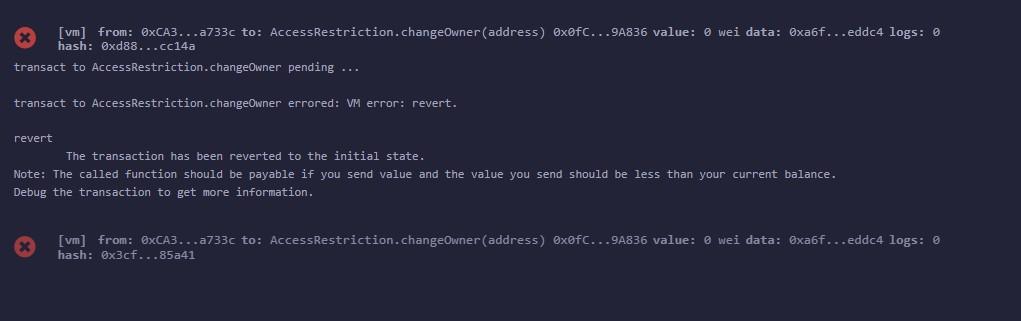
**Step 6->** You should get an error as following



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

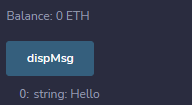


**Step 8->** Now, paste the actual address of the account in the changeowner input and click on changeowner



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

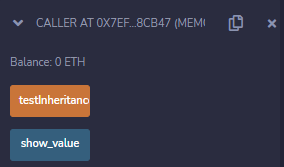
### Contracts



### Inheritance

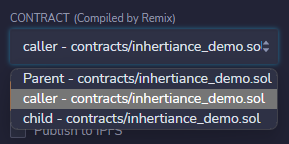




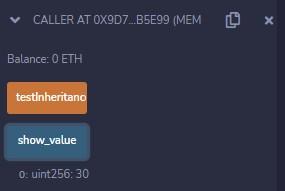


# Flow of execution

## **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

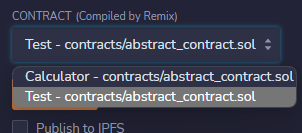


### Abstract Contracts

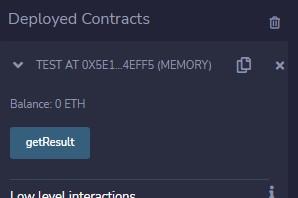


# Flow of execution

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



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

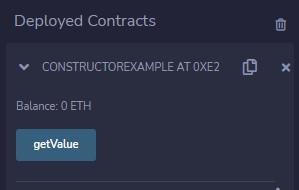


## **Step 3->** Click on getResult to get sum of a+b



### Constructors





# Flow of execution

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



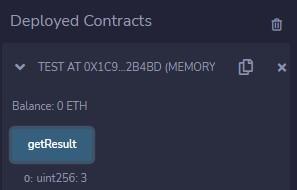
### Interfaces





# Flow of execution

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



## Libraries, Assembly, Events, Error handling.

### Libraries myLib.sol Code

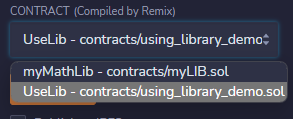
### using\_library.sol Code





# 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 output



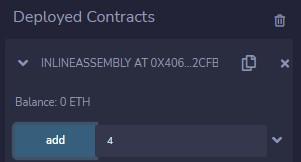
### Assembly





# Flow of execution

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



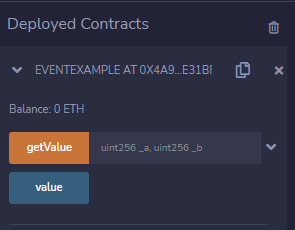
**Step 2->** Click add to output sum



### Events







# Flow of execution

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



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



### Error Handling

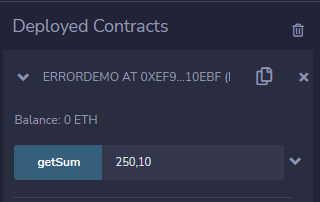




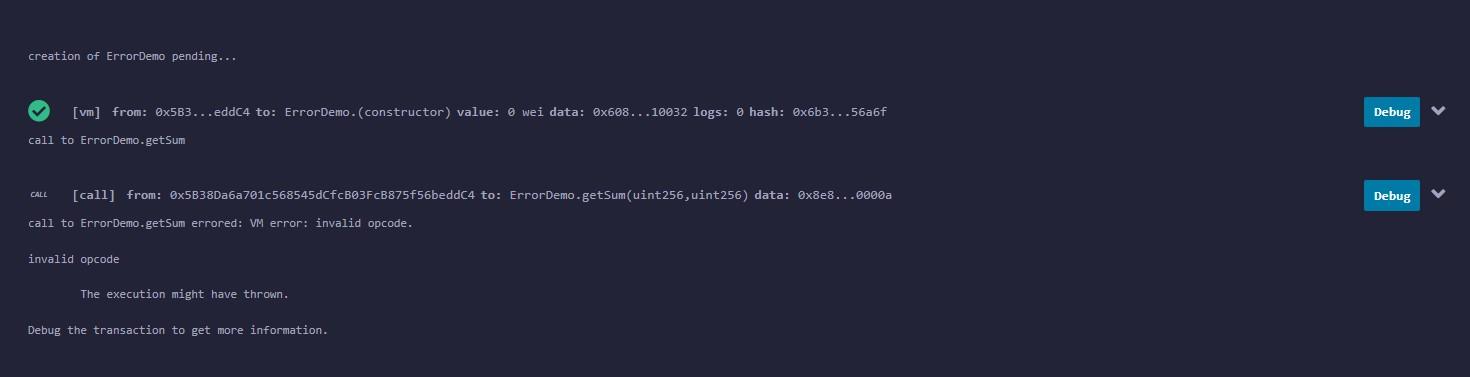


# Flow of execution

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



**Step 2->** Check terminal panel



# PRACTICAL-5 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);

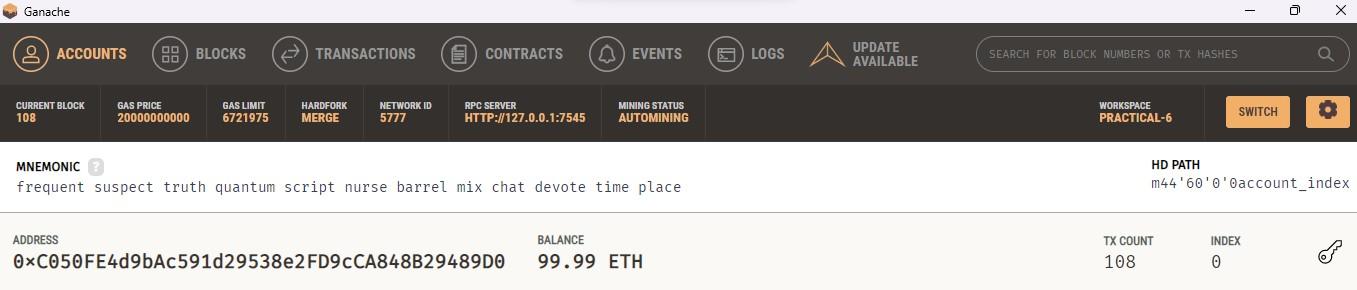
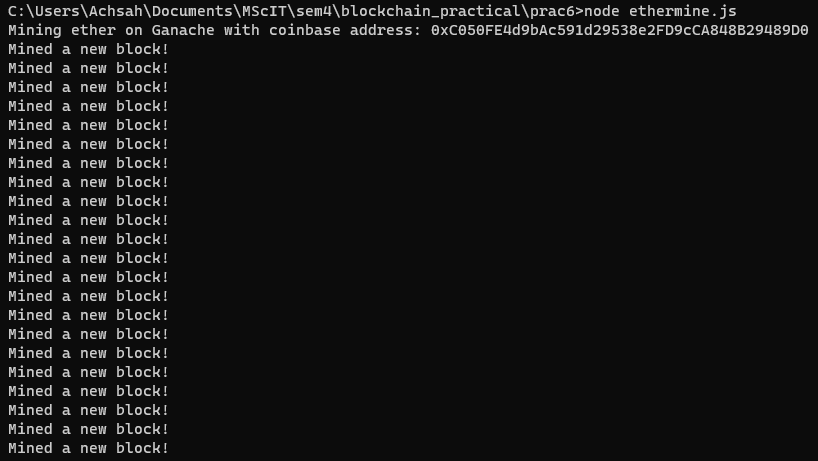
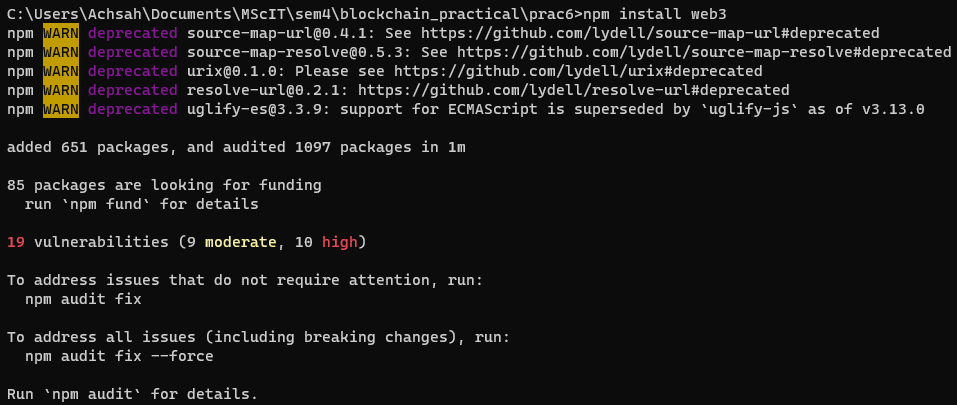
}

}

}

mine();





# PRACTICAL-6 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",

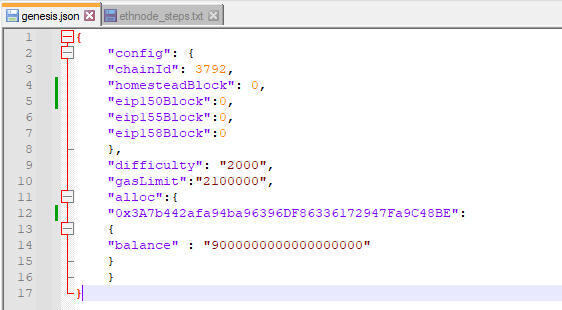
"gasLimit": "2100000", "alloc": {

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

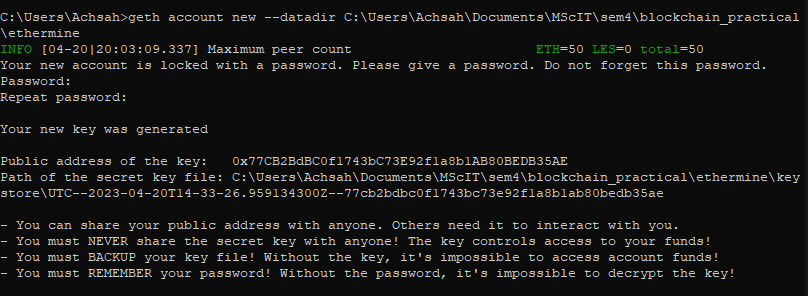
}

}

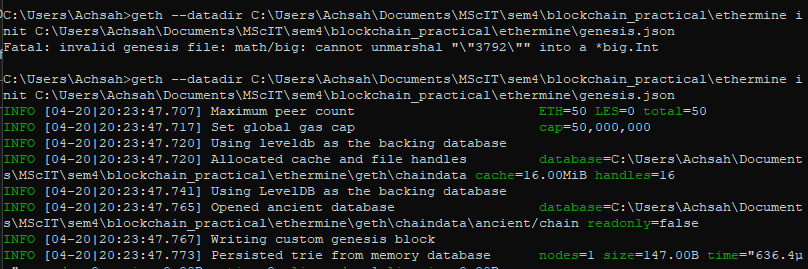
}



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



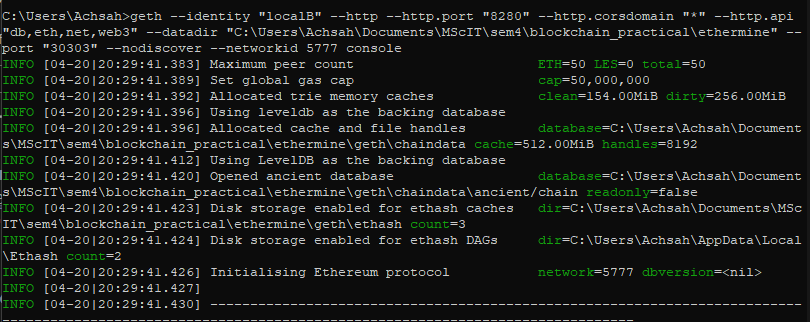
**Step 3->** Run command **geth account new --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain\_practical\ethermine**



# Step 4-> Run command geth --identity "localB" --http --http.port "8280"

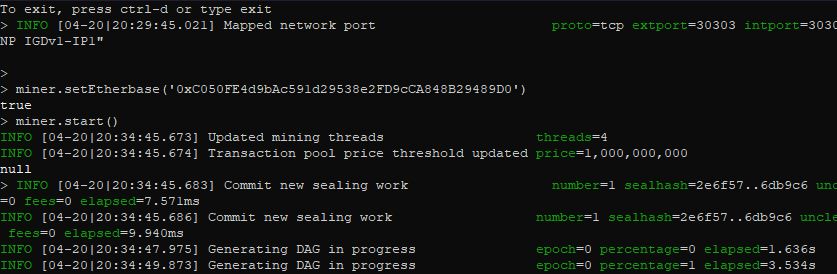
**--http.corsdomain "\*" --http.api "db,eth,net,web3" --datadir "C:\Users\Achsah\Documents\MScIT\sem4\blockchain\_practical\ethermine"**

**--port "30303" --nodiscover --networkid 5777 console**. This command will enable geth console.

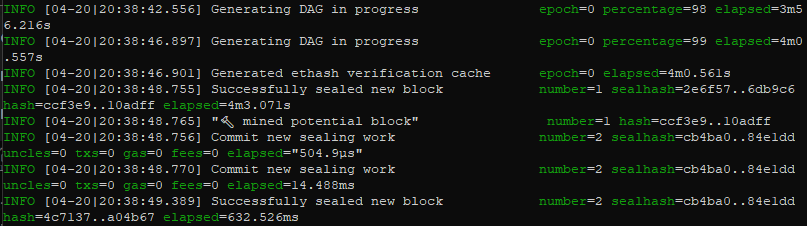


**Step 5->** Run the command **miner.setEtherbase('0xC050FE4d9bAc591d29538e2FD9cCA848B29489D0’)** in the geth console

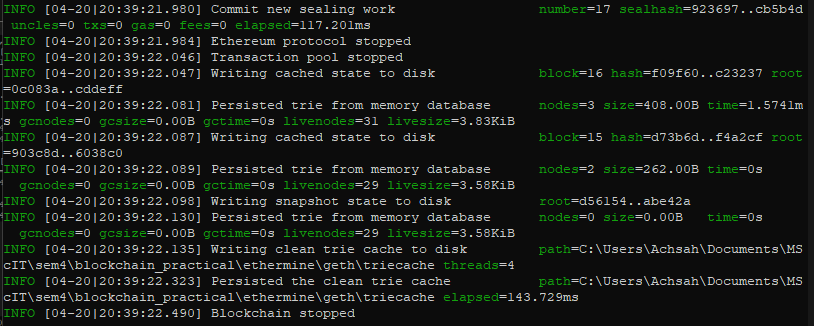
**Step 6->** Run the command **miner.start()** to start mining



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



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



# PRACTICAL-7 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));

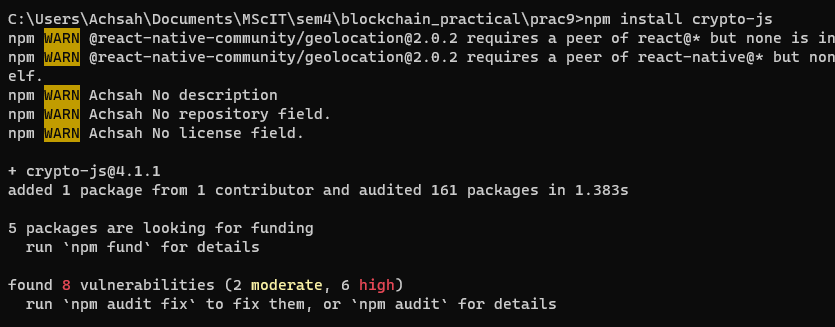


# Flow of execution

## **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

