Practical Journal

**BLOCKCHAIN**

Submitted by

Name: Rahul Chaurasia

SAP ID: 53004220018

Class: MSC IT PART II

**Department of Information Technology**

Shri Vile Parle Kelavani Mandal’s

**Usha Pravin Gandhi College of Arts, Science & Commerce**

(Affiliated to University of Mumbai)

NAAC Reaccredited ‘A+’ Grade

**Vile Parle (West), Mumbai 400056**

2023-24

**BLOCKCHAIN Practical Journal**

**Academic Year: 2023-2024**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INDEX** | | | | |
| **Sr. No.** | **Topic** | **Date** | **Page No** | **Sign** |
| **1.** | **Write the following programs for Blockchain in Python.** |  | **1** |  |
| A. | A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it. | 02/04/24 | 2 |  |
| B. | A transaction class to send and receive money and test it. | 02/04/24 |  |  |
| C. | Create multiple transactions and display them. | 02/04/24 |  |  |
| D. | Create a blockchain, a genesis block and execute it. | 09/04/24 |  |  |
| E. | Create a mining function and test it. | 09/04/24 |  |  |
| F. | Create a mining function and test it. | 09/04/24 |  |  |
| **2.** | **Implement and demonstrate the use of the following in Solidity.** |  | **5** |  |
| A. | Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables. | 16/04/24 | 6 |  |
| B. | Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions. | 25/04/24 |  |  |
| **3.** | **Implement and demonstrate the use of the following in Solidity.** |  | **10** |  |
| A. | Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces. | 25/04/24 | 11 |  |
| B. | Libraries, Assembly, Events, Error handling. | 07/05/24 |  |  |
| **4.** | **Write a program to demonstrate mining of Ether.** | 07/05/24 | **23** |  |
| **5.** | **Demonstrate the running of the blockchain node.** | 07/05/24 | **27** |  |
| **6.** | **Demonstrate the use of Bitcoin Core API.** | 15/05/24 | **33** |  |
| **7.** | **Create your own blockchain and demonstrate its use.** | 15/05/24 | **40** |  |
| **8.** | **Build D-Apps with angular.** | 15/05/24 | **45** |  |

**Practical 1**

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

**Code and Output:**

# pip install Crypto

# pip install pycryptodome

import math

# step 1

p = 3

q = 7

print("RSA Algorithm ")

# step 2

n = p\*q

print("n =", n)

# step 3

phi = (p-1)\*(q-1)

# step 4

e = 2 #change value

while(e<phi):

    if (math.gcd(e, phi) == 1):

        break

    else:

        e += 1

print("e =", e)

# step 5

k = 2

d = ((k\*phi)+1)/e

print("d =", d)

print(f'Public key: {e, n}')

print(f'Private key: {d, n}')

# plain text

msg = 15

print(f'Original message:{msg}')

# encryption

C = pow(msg, e)

C = math.fmod(C, n)

print(f'Encrypted message: {C}')

# decryption

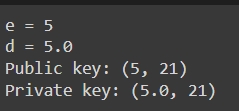
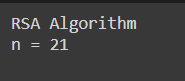
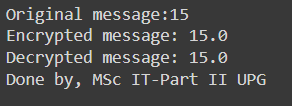
M = pow(C, d)

M = math.fmod(M, n)

print(f'Decrypted message: {M}')

print('Done by, MSc IT-Part II UPG')

**Output:**

****

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

**Code and Output:**

#Practical 1 B

# following imports are required by PKI

import hashlib

import random

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

    def \_\_init\_\_(self):

     random = 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')

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):

        if self.sender == "Genesis":

            identity = "Genesis"

        else:

            identity = 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')

def display\_transaction(transaction):

#for transaction in transactions:

    dict = transaction.to\_dict()

    print ("sender: " + dict['sender'])

    print ('-----')

    print ("recipient: " + dict['recipient'])

    print ('-----')

    print ("value: " + str(dict['value']))

    print ('-----')

    print ("time: " + str(dict['time']))

    print ('-----')

transactions = []

Dinesh = Client()

Ramesh = Client()

Suresh = Client()

t1 = Transaction( Dinesh, Ramesh.identity, 15.0)

t1.sign\_transaction()

transactions.append(t1)

t2 = Transaction( Ramesh, Suresh.identity, 25.0)

t2.sign\_transaction()

transactions.append(t2)

t3 = Transaction( Ramesh, Suresh.identity,200.0)

t3.sign\_transaction()

transactions.append(t3)

tn=1

for t in transactions:

  print("Transaction #",tn)

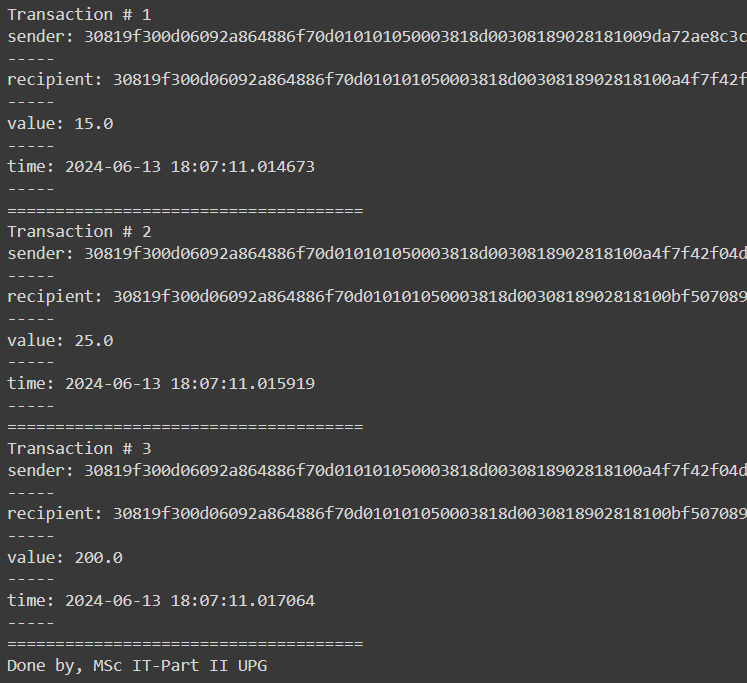
  display\_transaction (t)

  tn=tn+1

  print ('=====================================')

print('Done by, MSc IT-Part II UPG')

**Output**



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

**Code and Output:**

#Practical 1 C

# following imports are required by PKI

import hashlib

import random

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

    def \_\_init\_\_(self):

     random = 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')

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):

        if self.sender == "Genesis":

            identity = "Genesis"

        else:

            identity = 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')

def display\_transaction(transaction):

#for transaction in transactions:

    dict = transaction.to\_dict()

    print ("sender: " + dict['sender'])

    print ('-----')

    print ("recipient: " + dict['recipient'])

    print ('-----')

    print ("value: " + str(dict['value']))

    print ('-----')

    print ("time: " + str(dict['time']))

    print ('-----')

transactions = []

Kartik = Client()

Rakesh = Client()

Suraj = Client()

t1 = Transaction( Kartik, Rakesh.identity, 15.0)

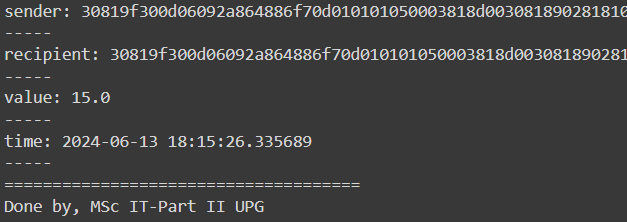
t1.sign\_transaction()

display\_transaction (t1)

print ('=====================================')

print('Done by, MSc IT-Part II UPG')

**Output**



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

**Code and Output:**

#Practical 1D

# following imports are required by PKI

import hashlib

import random

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

    def \_\_init\_\_(self):

     random = 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')

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):

        if self.sender == "Genesis":

            identity = "Genesis"

        else:

            identity = 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')

def display\_transaction(transaction):

#for transaction in transactions:

    dict = transaction.to\_dict()

    print ("sender: " + dict['sender'])

    print ('-----')

    print ("recipient: " + dict['recipient'])

    print ('-----')

    print ("value: " + str(dict['value']))

    print ('-----')

    print ("time: " + str(dict['time']))

    print ('-----')

def dump\_blockchain (self):

  print ("Number of blocks in the chain: " + str(len (self)))

  for x in range (len(TPCoins)):

    block\_temp = TPCoins[x]

    print ("block # " + str(x))

    for transaction in block\_temp.verified\_transactions:

         display\_transaction (transaction)

         print ('----------------------')

         print ('=====================================')

class Block:

         def \_\_init\_\_(self):

          self.verified\_transactions = []

          self.previous\_block\_hash = ""

          self.Nonce = ""

Dinesh = Client()

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

block0 = Block()

block0.previous\_block\_hash = None

Nonce = None

block0.verified\_transactions.append (t0)

digest = hash (block0)

last\_block\_hash = digest

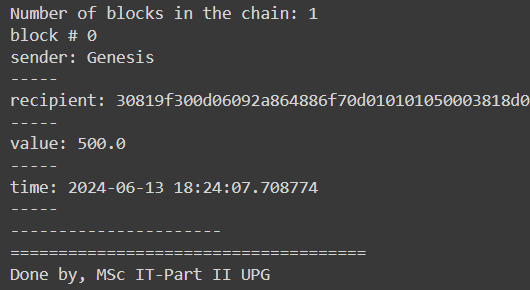
TPCoins = []

TPCoins.append (block0)

dump\_blockchain(TPCoins)

print('Done by, MSc IT-Part II UPG')

**Output**



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

**Code and Output:**

#Practical 1E

import hashlib

def sha256(message):

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

def mine(message, difficulty=1):

  assert difficulty >= 1

  #if(difficulty <1):

  # return #'1'\*2=> '11'

  prefix = '1' \* difficulty

  print("prefix",prefix)

  for i in range(1000):

    digest = sha256(str(hash(message)) + str(i))

    print("testing=>"+digest)

    if digest.startswith(prefix):

      print ("after " + str(i) + " iterations found nonce: "+ digest)

      return i #i= nonce value

mine ("Test message",2)

print('Done by, MSc IT-Part II UPG')

**Output**





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

**Code and Output:**

#Practical 1 F

# following imports are required by PKI

import hashlib

import random

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

    def \_\_init\_\_(self):

     random = 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')

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):

        if self.sender == "Genesis":

            identity = "Genesis"

        else:

            identity = 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')

def display\_transaction(transaction):

#for transaction in transactions:

    dict = transaction.to\_dict()

    print ("sender: " + dict['sender'])

    print ('-----')

    print ("recipient: " + dict['recipient'])

    print ('-----')

    print ("value: " + str(dict['value']))

    print ('-----')

    print ("time: " + str(dict['time']))

    print ('-----')

def dump\_blockchain (self):

    print ("Number of blocks in the chain: " + str(len (self)))

    for x in range (len(TPCoins)):

        block\_temp = TPCoins[x]

        print ("block # " + str(x))

        for transaction in block\_temp.verified\_transactions:

            display\_transaction (transaction)

            print ('--------------')

        print ('=====================================')

class Block:

    def \_\_init\_\_(self):

        self.verified\_transactions = []

        self.previous\_block\_hash = ""

        self.Nonce = ""

def sha256(message):

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

def mine(message, difficulty=1):

    assert difficulty >= 1

 #if(difficulty <1):

 # return

 #'1'\*3=> '111'

    prefix = '1' \* difficulty

    for i in range(1000):

        digest = sha256(str(hash(message)) + str(i))

    if digest.startswith(prefix):

        return i #i= nonce value

A = Client()

B =Client()

C =Client()

t0 = Transaction (

        "Genesis",

        A.identity,

        500.0)

t1 = Transaction (

        A,

        B.identity,

        40.0)

t2 = Transaction (

        A,

        C.identity,

        70.0)

t3 = Transaction (

        B,

        C.identity,

        700.0)

TPCoins = []

block0 = Block()

block0.previous\_block\_hash = None

Nonce = None

block0.verified\_transactions.append (t0)

digest = hash (block0)

last\_block\_hash = digest #last\_block\_hash it is hash of block0

TPCoins.append (block0)

block1 = Block()

block1.previous\_block\_hash = last\_block\_hash

block1.verified\_transactions.append (t1)

block1.verified\_transactions.append (t2)

block1.Nonce=mine (block1, 2)

digest = hash (block1)

last\_block\_hash = digest

TPCoins.append (block1)

block2 = Block()

block2.previous\_block\_hash = last\_block\_hash

block2.verified\_transactions.append (t3)

Nonce = mine (block2, 2)

block2.Nonce=mine (block2, 2)

digest = hash (block2)

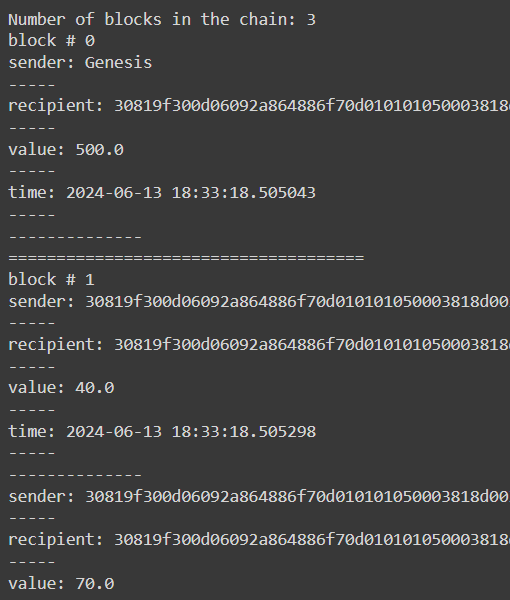
last\_block\_hash = digest

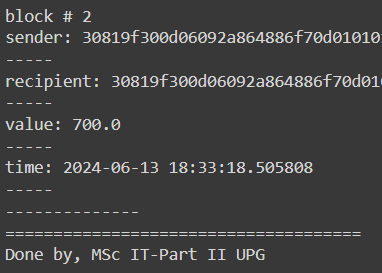
TPCoins.append (block2)

dump\_blockchain(TPCoins)

print('Done by, MSc IT-Part II UPG')

**Output**





**Practical 2**

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

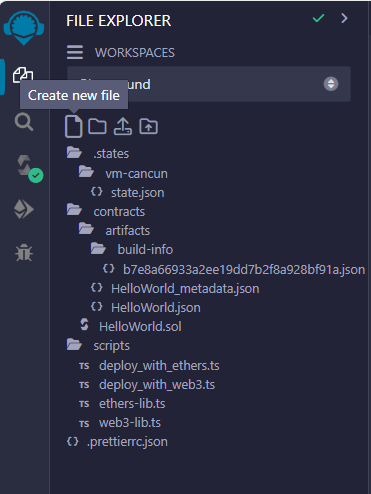
**Code and Output:**

**i. Variables**

1. **State Variable**

Step 1: Open <https://remix.ethereum.org/> website

Step 2: Create a new file and name it.



Step 3: Write this program in new file

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

contract SolidityTest{uint storedData; // State variable

constructor()  {

storedData = 10;

}

function getResult() public pure returns(uint){

uint a = 1; // local variable

uint b = 5;

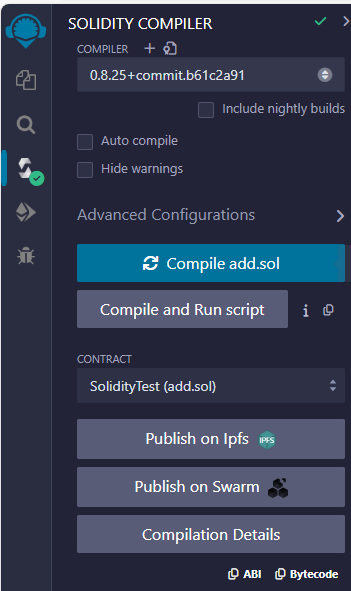
uint result = a + b;

return result; //access the state variable

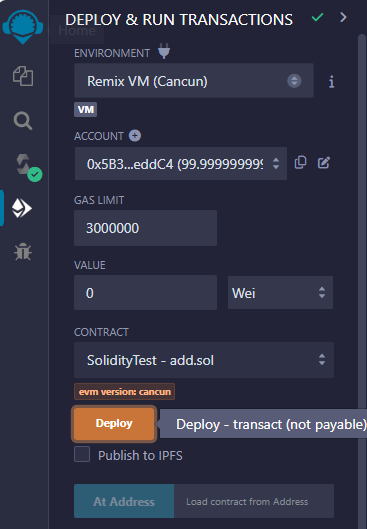
}

}

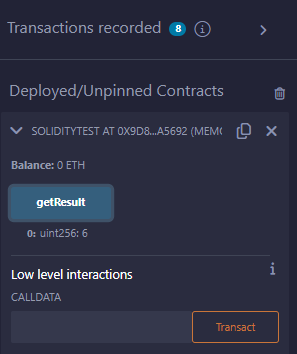
Step 4: Compile contract



Step 5: Deploy contract



Step 6: Select contract and click button getResult



1. **Local Variable**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

// Creating a contract

contract Solidity\_var\_Test {

// Defining function to show the declaration and

// scope of local variables

function getResult() public pure returns(uint){

    // Initializing local variables

    uint local\_var1 = 1;

    uint local\_var2 = 2;

    uint result = local\_var1 + local\_var2;

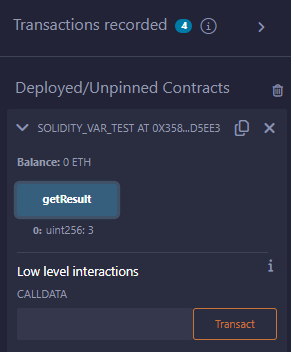
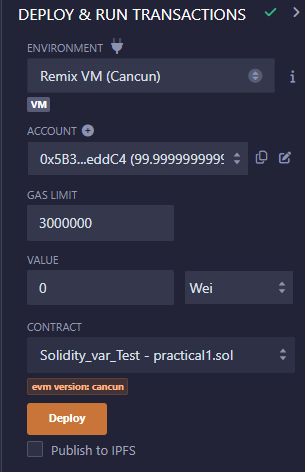
    // Access the local variable

    return result;

}

}

**Output:**



1. **Global Variable**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

// Creating a contract

contract Test {

    // Defining a variable

    address public admin;

    // Creating a constructor to use Global variable

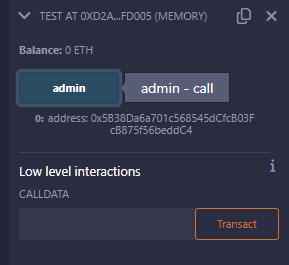
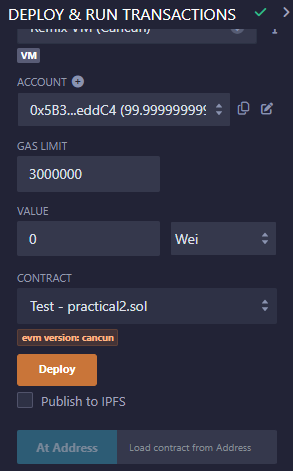
    constructor() {

        admin = msg.sender;

    }

}

**Output:**



**ii. Operators**

1. **Arithmetic Operator**

// SPDX-License-Identifier: MIT

pragma solidity <0.5.0;

// Creating a contract

contract SolidityTest {

// Initializing variables

uint16 public a = 20;

uint16 public b = 10;

// Initializing a variable with sum

uint public sum = a + b;

// Initializing a variable with the difference

uint public diff = a - b;

// Initializing a variable with product

uint public mul = a \* b;

// Initializing a variable with quotient

uint public div = a / b;

// Initializing a variable with modulus

uint public mod = a % b;

// Initializing a variabl decrement value

uint public dec = --b;

// Initializing a variable with increment value

uint public inc = ++a;

}

**Output:**

****

1. **Relational Operator**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;

// Creating a contract

contract SolidityTest {

// Declaring variables

uint16 public a = 50;

uint16 public b = 30;

// Initializing a variable with bool equal result

bool public eq = a == b;

// Initializing a variable with bool not equal result

bool public noteq = a != b;

// Initializing a variable with bool greater than result

bool public gtr = a > b;

// Initializing a variable with bool less than result

bool public les = a < b;

// Initializing a variable with bool greater than equal to result

bool public gtreq = a >= b;

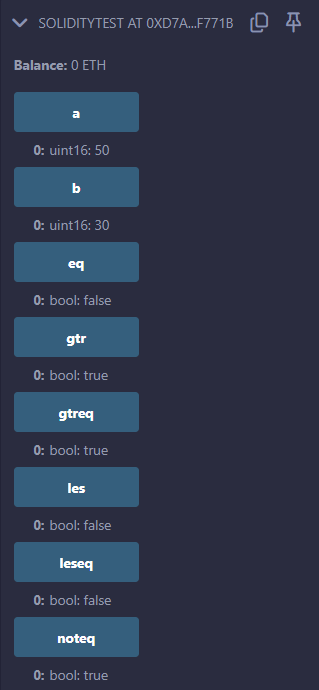
// Initializing a variable

// bool less than equal to result

bool public leseq = a <= b;

}

**Output:**

****

1. **Logical Operator**

**Code:**

pragma solidity ^0.5.0;

// Creating a contract

contract logicalOperator{

// Defining function to demonstrate Logical operator

function Logic(bool a, bool b) public view returns(

    bool, bool, bool){

        // Logical AND operator

        bool and = a&&b;

        // Logical OR operator

        bool or = a||b;

        // Logical NOT operator

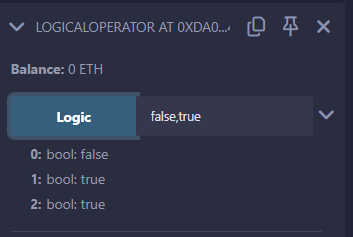
        bool not = !a;

        return (and, or, not);

    }

}

**Output:**

****

1. **Bitwise Operator**

**Code:**

pragma solidity ^0.5.0;

// Creating a contract

contract SolidityTest {

// Declaring variables

uint16 public a = 50;

uint16 public b = 70;

// Initializing a variable to '&' value

uint16 public and = a & b;

// Initializing a variable to '|' value

uint16 public or = a | b;

// Initializing a variable to '^' value

uint16 public xor = a ^ b;

// Initializing a variable to '<<' value

uint16 public leftshift = a << b;

// Initializing a variable to '>>' value

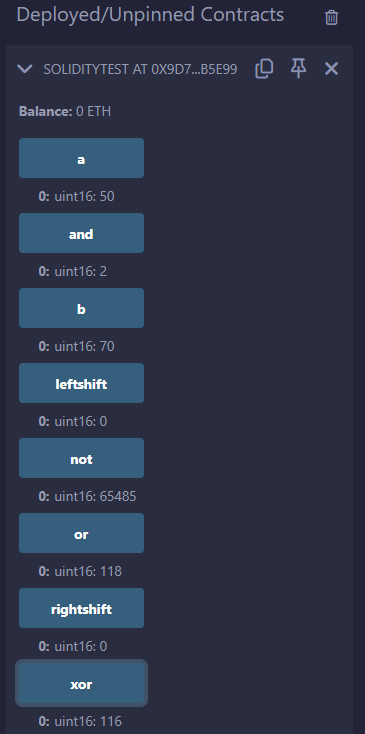
uint16 public rightshift = a >> b;

// Initializing a variable to '~' value

uint16 public not = ~a ;

}

**Output:**

****

1. **Assignment Operator**

**Code:**

pragma solidity ^0.5.0;

// Creating a contract

contract SolidityTest {

// Declaring variables

uint16 public assignment = 20;

uint public assignment\_add = 50;

uint public assign\_sub = 50;

uint public assign\_mul = 10;

uint public assign\_div = 50;

uint public assign\_mod = 32;

// Defining function to demonstrate Assignment Operator

function getResult() public{

    assignment\_add += 10;

    assign\_sub -= 20;

    assign\_mul \*= 10;

    assign\_div /= 10;

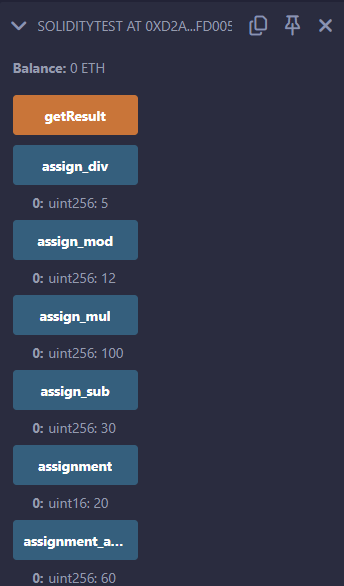
    assign\_mod %= 20;

    return ;

}

}

**Output:**

****

1. **Conditional Operator**

**Code:**

pragma solidity ^0.5.0;

// Creating a contract

contract SolidityTest{

// Defining function to demonstrate conditional operator

function sub(

uint a, uint b) public view returns( uint){

    uint result = (a > b? a-b : b-a);

    return result;

}

}

**Output:**

****

**iii. Loops**

1. **While Loop**

**Code:**

pragma solidity ^0.5.0;

contract SolidityTest { uint storedData; constructor() public{ storedData = 10;

}

function getResult() public view returns(string memory){

    uint a = 10;

    uint b = 2;

    uint result = a + b;

    return integerToString(result);

}

function integerToString(uint \_i) internal pure returns (string memory) {

    if (\_i == 0) { return "0";

}

uint j = \_i; uint len;

while (j != 0) {

    len++;

    j /= 10;

}

bytes memory bstr = new bytes(len); uint k = len - 1;

while (\_i != 0) {

    bstr[k--] = byte(uint8(48 + \_i % 10));

    \_i /= 10;

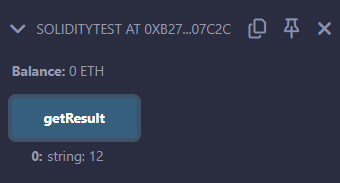
}

return string(bstr);

}

}

**Output:**

****

1. **Do-while Loop**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract SolidityTest {

uint storedData;

constructor() public{

storedData = 10;

}

function getResult() public view returns(string memory){

uint a = 10;

uint b = 2;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {

return "0";

}

uint j=0;

uint len;

for (j = \_i; j != 0; j /= 10) {  //for loop example

len++;

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

bstr[k--] = byte(uint8(48 + \_i % 10));

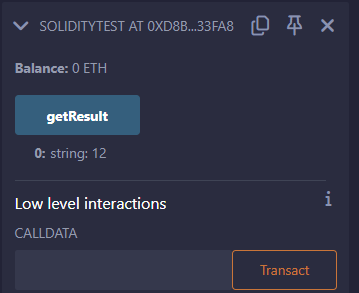
\_i /= 10;

}

return string(bstr);//access local variable

}}

**Output:**



1. **For Loop**

**Code and Output:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract SolidityTest {

uint storedData;

constructor() public{

storedData = 10;

}

function getResult() public view returns(string memory){

uint a = 10;

uint b = 2;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {

return "0";

}

uint j=0;

uint len;

for (j = \_i; j != 0; j /= 10) {  //for loop example

len++;

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

bstr[k--] = byte(uint8(48 + \_i % 10));

\_i /= 10;

}

return string(bstr);//access local variable

}}

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract SolidityTest {

uint storedData;

constructor() public{

storedData = 10;

}

function getResult() public view returns(string memory){

uint a = 10;

uint b = 2;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {

return "0";

}

uint j=0;

uint len;

for (j = \_i; j != 0; j /= 10) {  //for loop example

len++;

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

bstr[k--] = byte(uint8(48 + \_i % 10));

\_i /= 10;

}

return string(bstr);//access local variable

}}

**4. loop Control: (Break statement)**

**Code and Output:**

I)BREAK

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract SolidityTest {

uint storedData;

constructor() public{

storedData = 10;

}

function getResult() public view returns(string memory){

uint a = 1;

uint b = 2;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {

return "0";

}

uint j = \_i;

uint len;

while (true) {

len++;

j /= 10;

if(j==0){

break;   //using break statement

}

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

bstr[k--] = byte(uint8(48 + \_i % 10));

\_i /= 10;

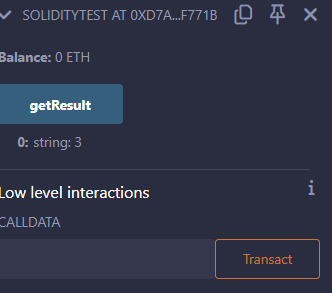
}

return string(bstr);

}

}

**Output**



**5. Continue statement**

**Code and Output:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract SolidityTest {

uint storedData;

constructor() public{

storedData = 10;

}

function getResult() public view returns(string memory){

uint n = 1;

uint sum = 0;

while( n < 10){

n++;

if(n == 5){

continue; // skip n in sum when it is 5.

}

sum = sum + n;

}

return integerToString(sum);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {

return "0";

}

uint j = \_i;

uint len;

while (true) {

len++;

j /= 10;

if(j==0){

break;   //using break statement

}

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

bstr[k--] = byte(uint8(48 + \_i % 10));

\_i /= 10;

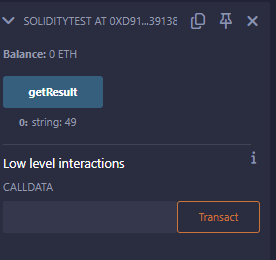
}

return string(bstr);

}

}

**Output:**



**iv. Decision Making:**

**1. If-statement**

**Code and Output:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract SolidityTest {

uint storedData;

constructor() public {

storedData = 10;

}

function getResult() public view returns(string memory){

uint a = 1;

uint b = 2;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {   // if statement

return "0";

}

uint j = \_i;

uint len;

while (j != 0) {

len++;

j /= 10;

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

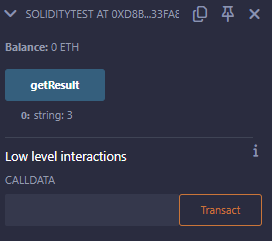
bstr[k--] = byte(uint8(48 + \_i % 10));

\_i /= 10;

}

return string(bstr);//access local variable

}}



**2.If-else statement:**

**Code and Output:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract Types {

// Declaring state variables

uint i = 10;

bool even;

// Defining function to

// demonstrate the use of

// 'if...else statement'

function decision\_making(

) public payable returns(bool){

if (i%2 == 0){

even = true;

}

else{

even = false;

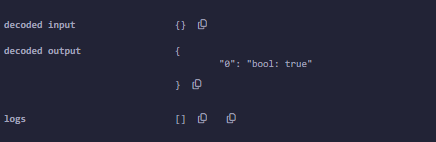
}

return even;

}

}

**Output:**



**3.If-else..if statement:**

**Code and Output:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.12 <0.8.0;

contract Types {

// Declaring state variables

uint i = 12;

string result;

// Defining function to

// demonstrate the use

// of 'if...else if...else

// statement'

function decision\_making (

) public returns(string memory){

if(i<10){

result = "less than 10";

}

else if(i == 10){

result = "equal to 10";

}

else{

result = "greater than 10";

}

return result;

}

}

**Output:**



**v. String**

**Code and Output:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

// Creating a contract

contract Test {

// Declaring variable

string  str;

// Defining a constructor

constructor(string memory str\_in){

    str = str\_in;

}

// Defining a function to

// return value of variable 'str'

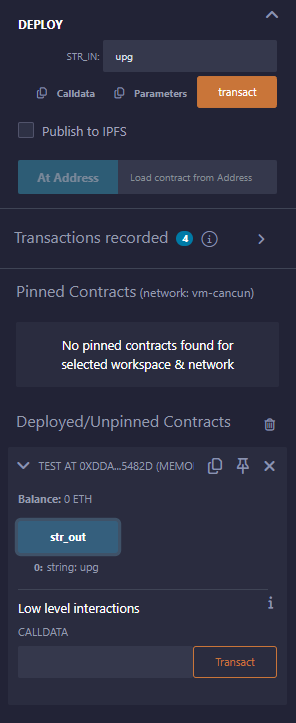
function str\_out() public view returns(string memory){

return str;

}

}

**Output:**



**vi. Arrays:**

**Code and Output:**

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.12 <0.9.0;

// Creating a contract

contract Types {

// Declaring an array

uint[6] data;

uint x;

// Defining function to

// assign values to array

function array\_example() public returns (uint[6] memory)

{

data  = [uint(10), 20, 30, 40, 50, 60];

}

function result() public view returns(uint[6] memory){

return data;

}

// Defining function to access

// values from the array

// from a specific index

function array\_element() public view returns (uint){

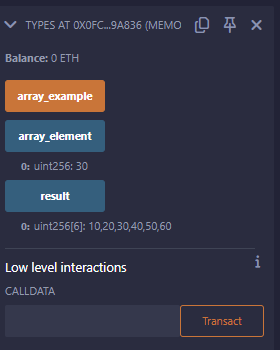
uint x = data[2];

return x;

}

}

**Output:**



**vii. Enums**

**Code and Output:**

pragma solidity >=0.5.12 <0.8.0;

// Creating a contract

contract Types

{

// Creating an enumerator

enum week\_days {

Monday

,

Tuesday

,

Wednesday

,

Thursday

,

Friday

,

Saturday

,

Sunday }

// Declaring variables of

// type enumerator

week\_days week

;

week\_days choice

;

// Setting a default value

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

// Defining a function to

// set value of choice

function set\_value() public

{

choice

= week\_days

.Thursday

;

}

// Defining a function to

// return value of choice

function get\_choice

(

) public view returns

(week\_days

)

{

return choice

;

}

// Defining function to

// return default value

function getdefaultvalue

(

) public pure returns

(week\_days

)

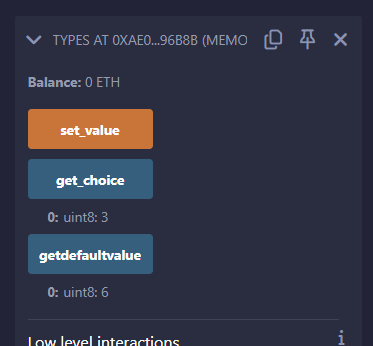
{

return default\_value

;

}}

**Output:**



**viii. Structure**

**Code and Output:**

pragma solidity >=0.5.12 <0.8.0;

contract test {

struct Book {

string title;

string author;

uint book\_id;

}

Book book;

function setBook() public {

book = Book('Learn Java', 'TP', 1);

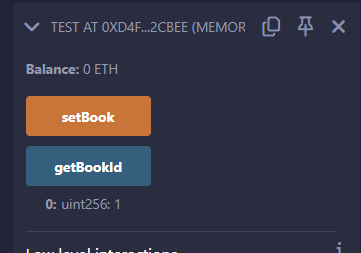
}

function getBookId() public view returns (uint) {

return book.book\_id;

}

}



**ix. Mappings**

**Code and Output:**

pragma solidity >=0.5.12 <0.8.0;

contract LedgerBalance {

mapping(address => uint) balance;

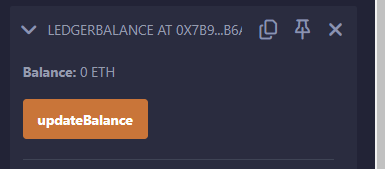
function updateBalance() public returns(uint) {

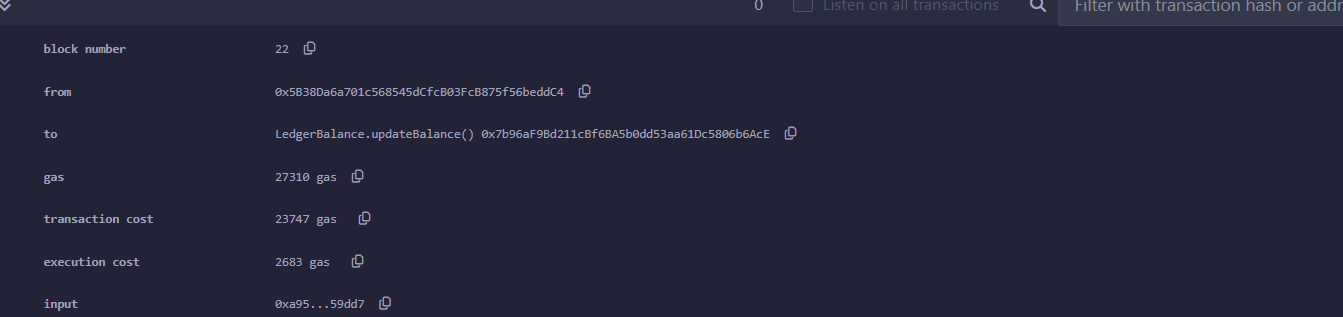
balance[msg.sender]=30;

return balance[msg.sender];

}

}





**Code:**

pragma solidity >=0.5.12 <0.8.0;

contract LedgerBalance {

mapping(address => string) name;

function updateBalance() public returns(string memory){

name[msg.sender] = "Mrunali";

return name[msg.sender];

}

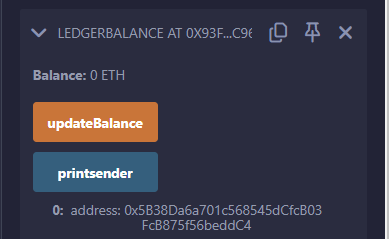
function printsender() public view returns(address) {

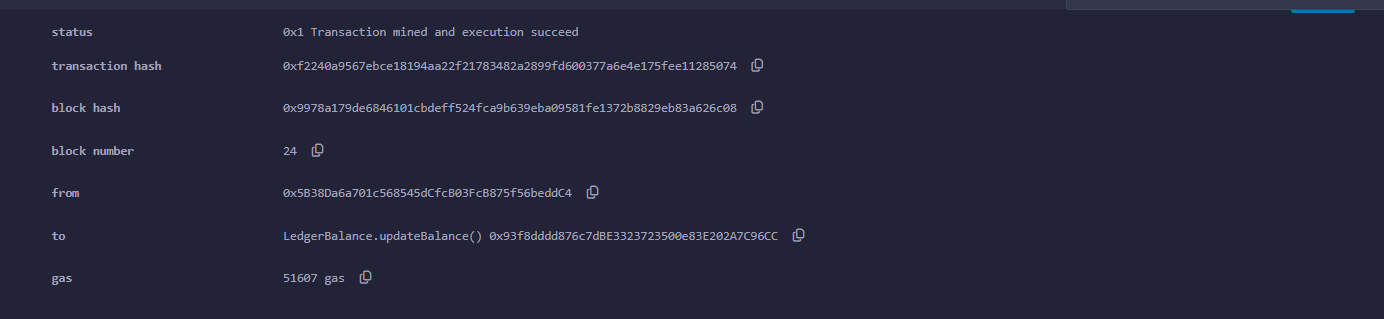
return msg.sender;

}

}

**Output**:





**Practical 3B**

**A) Function**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

contract SolidityTest {

constructor() public{

}

function getResult() public view returns(string memory){

uint a = 1;

uint b = 2;

uint result = a + b;

return integerToString(result);

}

function integerToString(uint \_i) internal pure

returns (string memory) {

if (\_i == 0) {

return "0";

}

uint j = \_i;

uint len;

while (j != 0) {

len++;

j /= 10;

}

bytes memory bstr = new bytes(len);

uint k = len - 1;

while (\_i != 0) {

bstr[k--] = byte(uint8(48 + \_i % 10));

\_i /= 10;

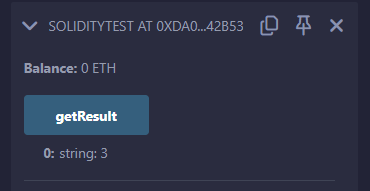
}

return string(bstr); //access local variable

}

}

**Output:**



**B) View Function:**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

contract Test {

function getResult() public view returns(uint product, uint sum){

uint a = 1; // local variable

uint b = 2;

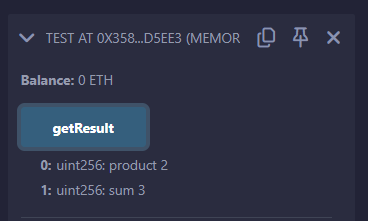
product = a \* b;

sum = a + b;

}

}

**Output:**



**C) Pure function:**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

contract Test {

function getResult() public pure returns(uint product, uint sum){

uint a = 12;

uint b = 5;

product = a \* b;

sum = a + b;

}

}

**Output:**



**D) Fallback Function**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

contract LedgerBalance {

string public calledFallbackFun;

fallback() external payable{

calledFallbackFun="Fallback function is executed!";

}

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

contract Sender

{

function transferEther() public payable

{

(bool sent, ) = payable(0xD4Fc541236927E2EAf8F27606bD7309C1Fc2cbee).call{value: 2 ether}("Transaction Completed!");

require(sent, "Transaction Failed!");

}

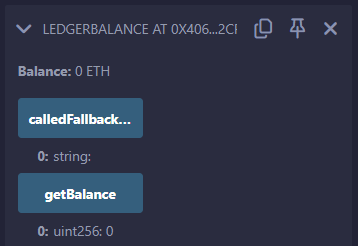
function getBalance() public view returns (uint) {

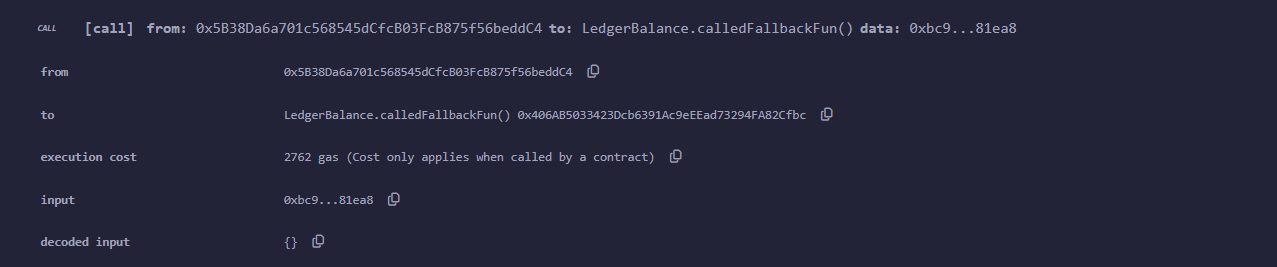
return address(this).balance;

}

}

**Output:**





**Practical #B (Contd)**

**A)Function Overloading**

**Code:**

contract Test {

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

return a + b;

}

function getSum(uint a, uint b, uint c ) public pure returns(uint){

return a + b + c;

}

function callSumWithTwoArguments() public pure returns(uint){

return getSum(2,2);

}

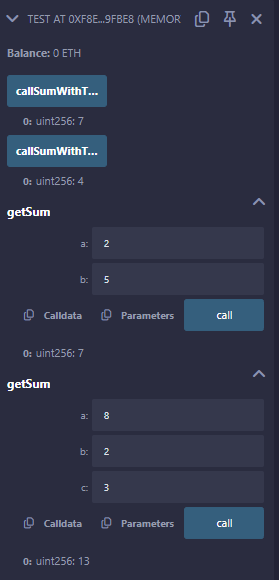
function callSumWithThreeArguments() public pure returns(uint){

return getSum(1,2,4);

}

}

**Output:**



**B)Mathematical Function**

**Code:**

contract Test {

function callAddMod() public pure returns(uint){

return addmod(4, 5, 3);

}

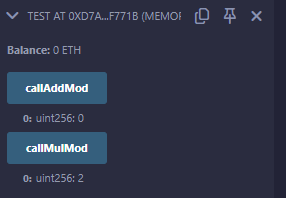
function callMulMod() public pure returns(uint){

return mulmod(4, 5, 3);

}

}

**Output:**



**C)Cryptographic Function**

contract Test {

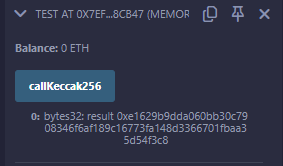
function callKeccak256() public pure returns(bytes32 result){

return keccak256("ABC");

}

}

**Output:**



**Practical 3**

**A. Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces.**

**i. Contracts**

pragma solidity >=0.5.0 <0.7.0;

contract C {

//private state variable

uint private data;

//public state variable

uint public info;

//constructor

constructor() public {

info = 10;

}

//private function

function increment(uint a) private pure returns(uint) { return a + 1; }

//public function

function updateData(uint a) public { data = a; }

function getData() virtual public view returns(uint) { return data; }

function compute(uint a, uint b) internal pure returns (uint) { return a + b; }

}

//Derived Contract

contract E is C {

uint private result;

C private c;

constructor() public {

c = new C();

}

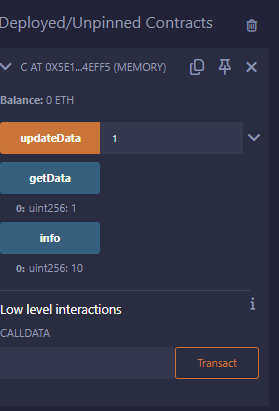
function getComputedResult() public {

result = compute(3, 5);

}

function getResult() public view returns(uint) { return result; }

funct



**Inheritance:**

pragma solidity >=0.5.0 <0.7.0;

// Defining contract

contract parent{

// Declaring internal

// state variable

uint internal sum;

// Defining external function

// to set value of internal

// state variable sum

function setValue() external {

uint a = 20;

uint b = 20;

sum = a + b;

}

}

// Defining child contract

contract child is parent{

// Defining external function

// to return value of

// internal state variable sum

function getValue() external view returns(uint) {

return sum;

}

}

// Defining calling contract

contract caller {

// Creating child contract object

child cc = new child();

// Defining function to call

// setValue and getValue functions

function testInheritance() public {

cc.setValue();

}

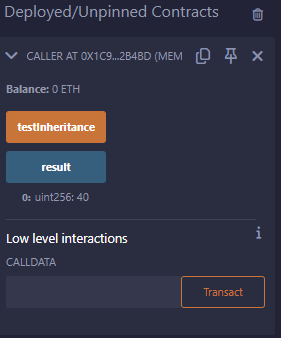
function result() public view returns(uint ){

return cc.getValue();

}

}

**Output:**



**Constructors:**

pragma solidity >=0.5.0 <0.7.0;

contract Base {

uint data;

constructor(uint \_data) public {

data = \_data;

}

function getresult()public view returns(uint){

return data;

}

}

contract Derived is Base (5) {

constructor() public {}

}

// Indirect Initialization of Base Constructor

pragma solidity >=0.5.0 <0.7.0;

contract Base {

uint data;

constructor(uint \_data) public {

data = \_data;

}

function getresult()public view returns(uint){

return data;

}

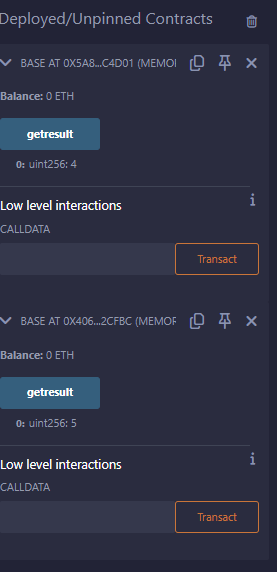
}

contract Derived is Base {

constructor(uint \_info) Base(\_info \* \_info) public {}

}

**Output:**



**D)Abstract Contracts:**

pragma solidity ^0.8.0;

abstract contract Calculator {

function getResult() public view virtual  returns(uint);

}

contract Test is Calculator {

function getResult() public view override returns(uint) {

uint a = 4;

uint b = 2;

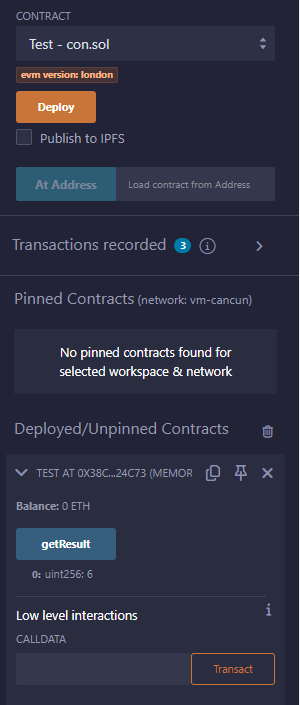
uint result = a + b;

return result;

}

}

**Output:**



**Interfaces:**

pragma solidity ^0.8.0;

interface Calculator {

function getResult() external view returns(uint);

}

contract Test is Calculator {

constructor() public {}

function getResult() external view returns(uint){

uint a = 5;

uint b = 2;

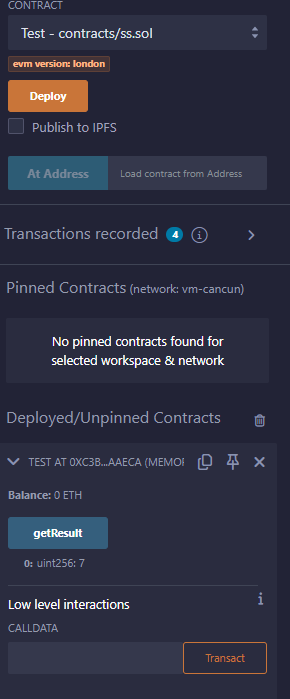
uint result = a + b;

return result;

}

}

**Output:**



**F) Libraries:**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

library Search {

function indexOf(uint[] storage self, uint value) public view returns (uint) {

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

if (self[i] == value) return i;

return uint(-1);}

}

contract Test {

uint[] data;

uint value;

uint index;

constructor() public {

data.push(6);

data.push(7);

data.push(8);

data.push(9);

data.push(10);

}

function isValuePresent() external {

value = 9;

//search if value is present in the array using Library function

index = Search.indexOf(data, value);

}

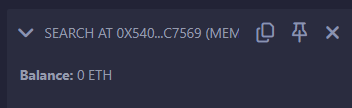
function getresult() public view returns(uint){

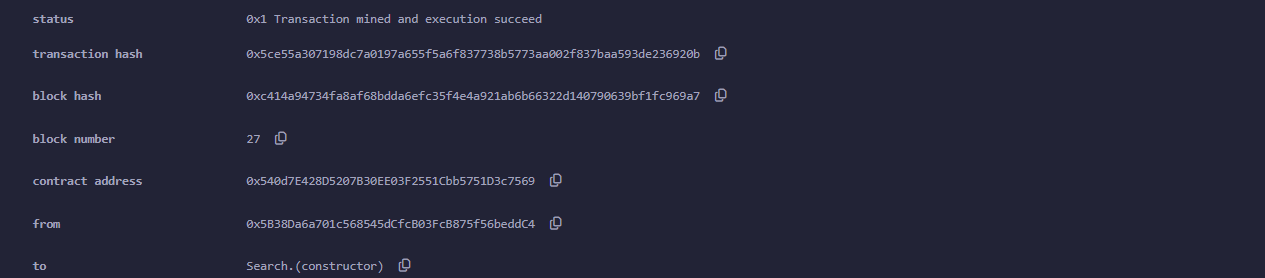
return index;

}

}

**Output:**





**G) Assembly:**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

library Sum {

function sumUsingInlineAssembly(uint[] memory \_data) public pure returns (uint o\_sum) {

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

assembly {

o\_sum := add(o\_sum, mload(add(add(\_data, 0x20), mul(i, 0x20))))

}}

}

}

contract Test {

uint[] data;

constructor() public {

data.push(1);

data.push(2);

data.push(3);

data.push(4);

data.push(5);

}

function sum() external view returns(uint){

return Sum.sumUsingInlineAssembly(data);

}

}

**Output:**



**H) Events:**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

// 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(uint \_a, uint \_b) public {

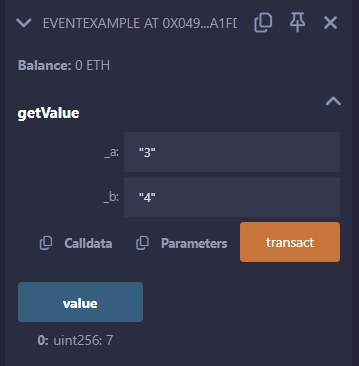
emit Increment(msg.sender);

value = \_a + \_b;

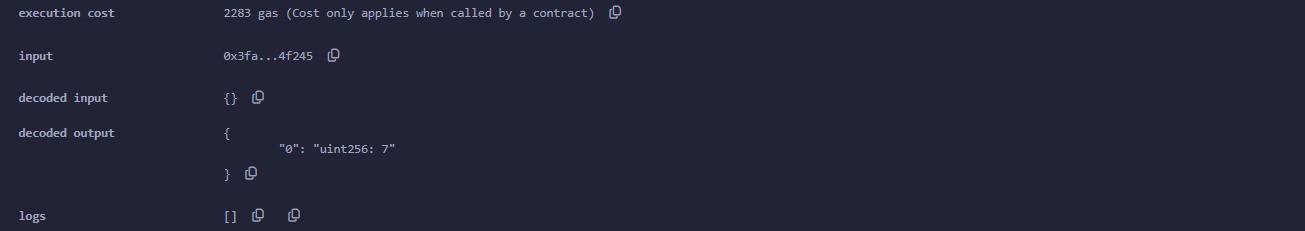
}

}

**Output:**







**I) Error Handling**

**Code:**

pragma solidity >=0.5.12 <0.8.0;

// Creating a contract

contract requireStatement {

// Defining function to

// check input

function checkInput(uint8 \_input) public view returns(string memory){

require(\_input >= 0, "invalid uint");

require(\_input <= 255, "invalid uint8");

return "Input is Uint8";

}

// Defining function to

// use require statement

function Odd(uint \_input) public view returns(bool){

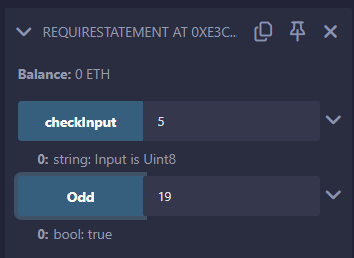
require(\_input % 2 != 0);

return true;

}

}

**Output:**



**Code:**

pragma solidity >=0.5.12 <0.8.0;

// Creating a contract

contract assertStatement {

// Defining a state variable

bool result;

// Defining a function

// to check condition

function checkOverflow(uint8 \_num1, uint8 \_num2) public {

uint8 sum = \_num1 + \_num2;

assert(sum<=255);

result = true;

}

// Defining a function to

// print result of assert

// statement

function getResult() public view returns(string memory){

if(result == true){

return "No Overflow";

}

else{

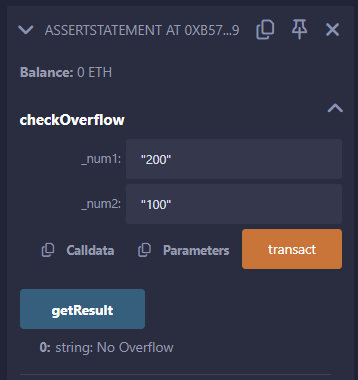
return "Overflow exist";

}

}

}

**Output:**





**Code:**

pragma solidity >=0.5.12 <0.8.0;

// Creating a contract

contract revertStatement {

// Defining a function

// to check condition

function checkOverflow(uint \_num1, uint \_num2) public view returns(

string memory, uint) {

uint sum = \_num1 + \_num2;

if(sum < 0 || sum > 255){

revert(" Overflow Exist");

}

else{

return ("No Overflow", sum);

}

}

}

**Output:**

**Practical 4**

**A. Write a program to demonstrate mining of Ether.**

**Code and Output:**

#Practical 4

import hashlib

def mine\_block(previous\_hash, transactions, difficulty):

  nonce = 0

  prefix = '0' \* difficulty

  while True:

    data = str(nonce) + previous\_hash + transactions

    hash\_result = hashlib.sha256(data.encode()).hexdigest()

    if hash\_result.startswith(prefix):

      print("Block mined successfully!")

      print("Nonce:", nonce)

      print("Hash:", hash\_result)

      return hash\_result

    nonce += 1

def main():

  previous\_hash ='0000000000000000000000000000000000000000000000000000000000000000' # Initial hash

  transactions = 'A sends 1 Ether to B' # Example transaction data

  difficulty = 5 # Number of leading zeros required in the hash

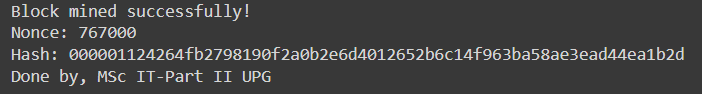
  mined\_hash = mine\_block(previous\_hash, transactions, difficulty)

  # Add code here to broadcast the mined hash to the Ethereum network

if \_\_name\_\_ == '\_\_main\_\_':

  main()

print('Done by, MSc IT-Part II UPG')

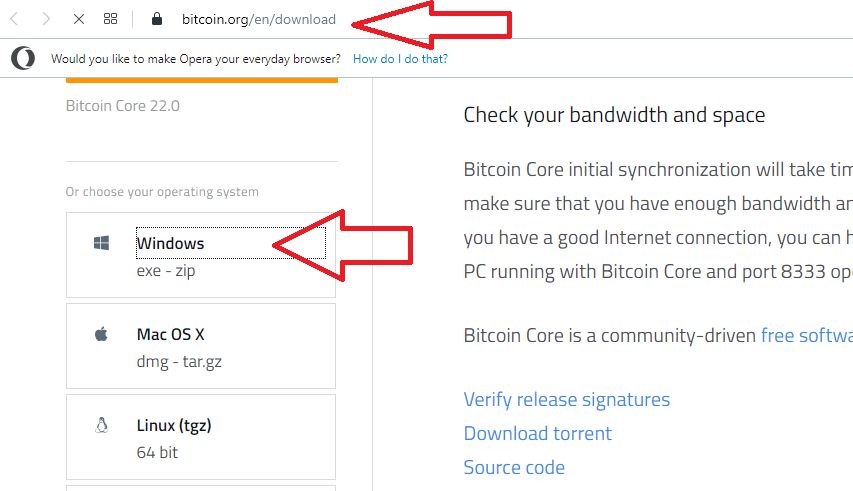


**Practical 5**

**A. Demonstrate the running of the blockchain node.**

Step 1: Visit: <https://bitcoin.org/en/download>

Step 2: Download windows setup [use and try with Linux version as well]

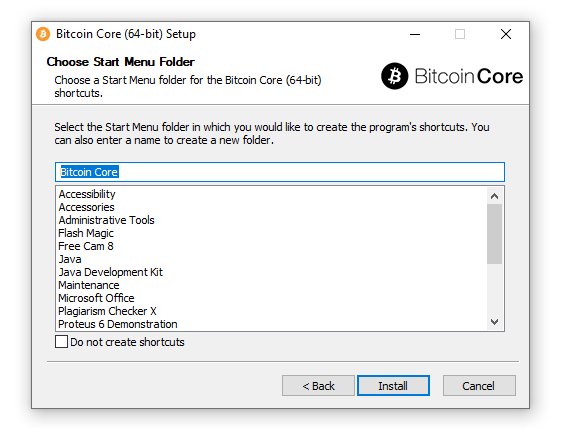


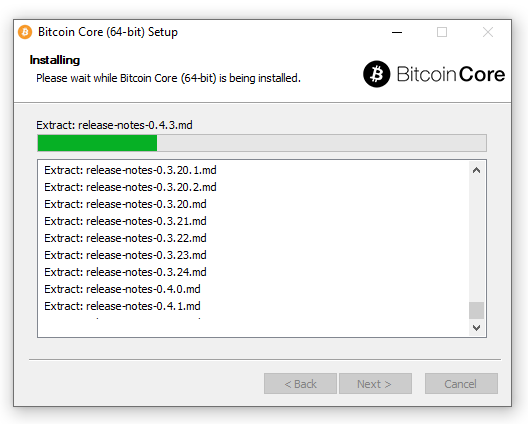
Step 3: Run the setup file-> click next

Step 4: Click Next



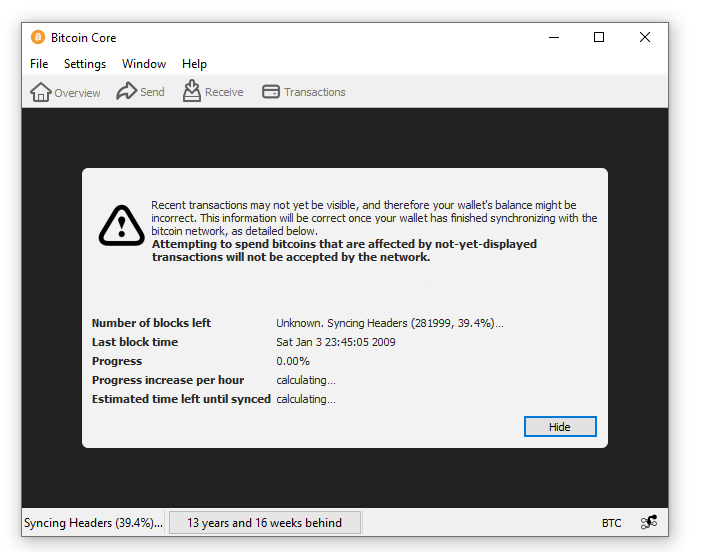
Step 5: Finally click on Install



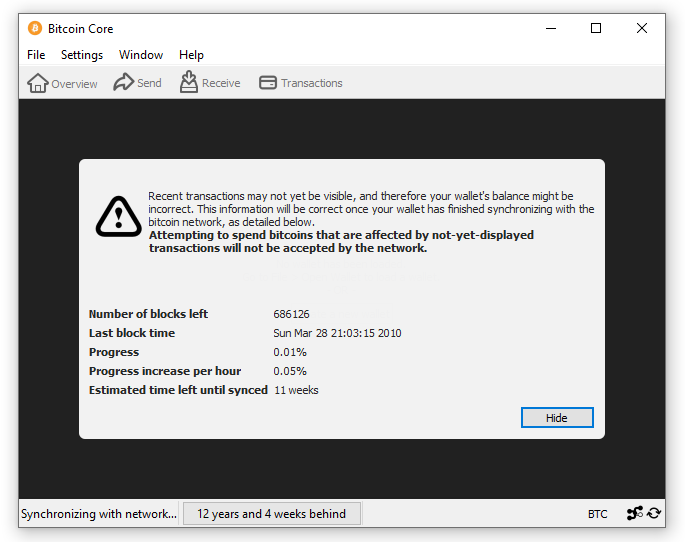


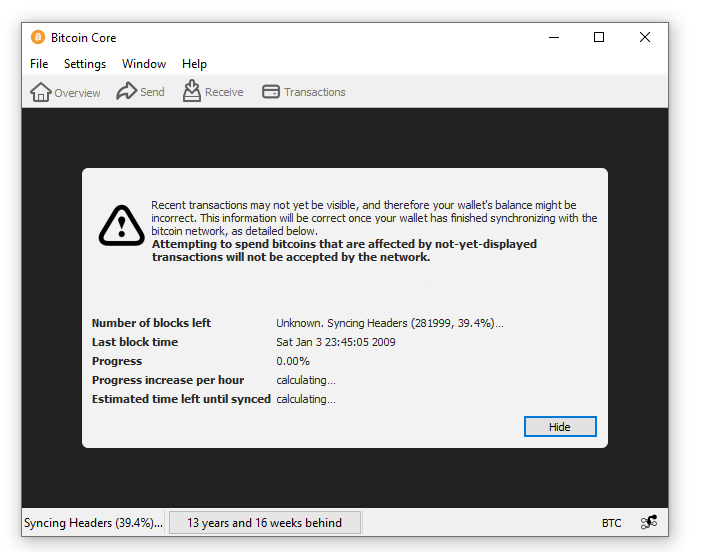
Step 6: Launch Bitcoin Core-> Click OK.



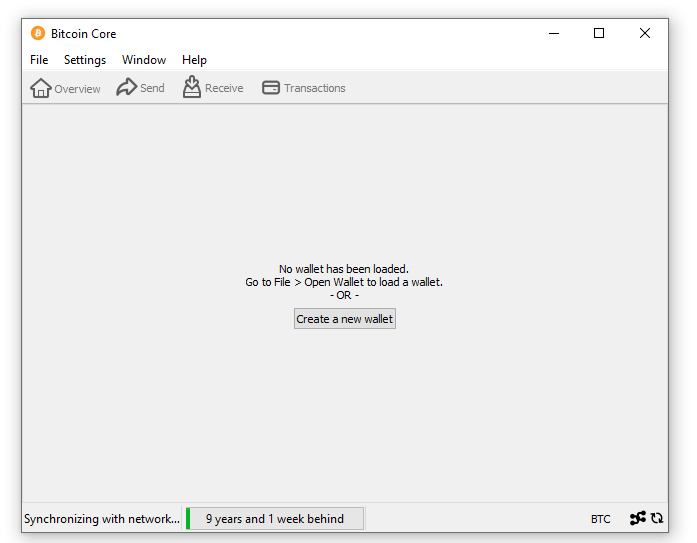


Step 7: Click on Hide button [Synchronization take place in background]

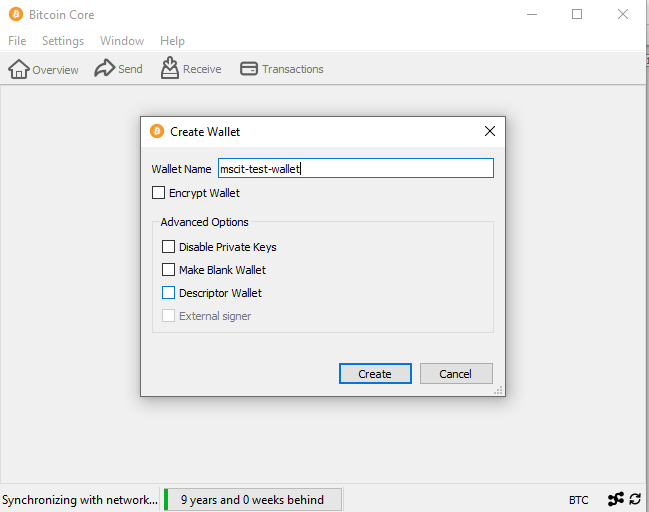




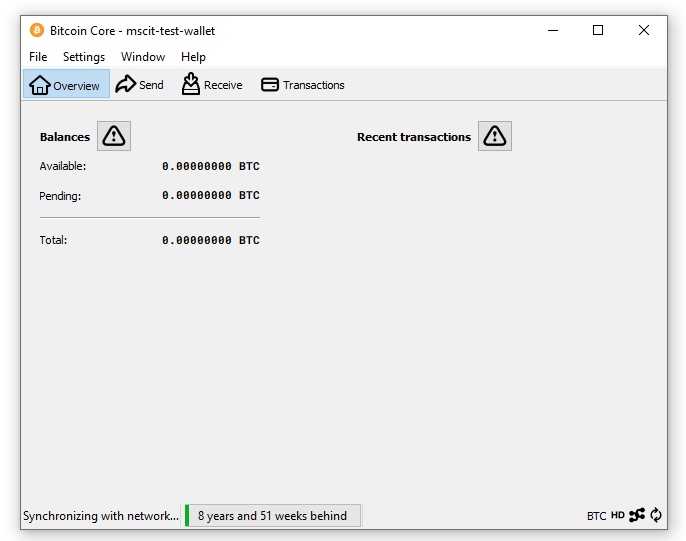
Step 8: You can create a wallet -> Create a new wallet

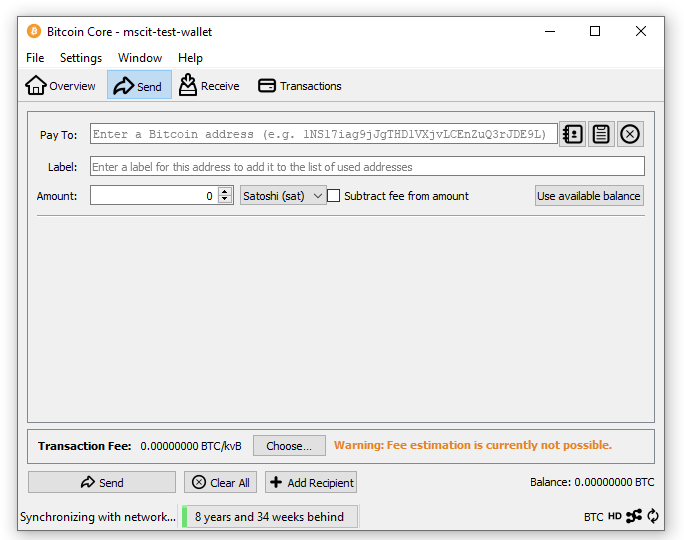


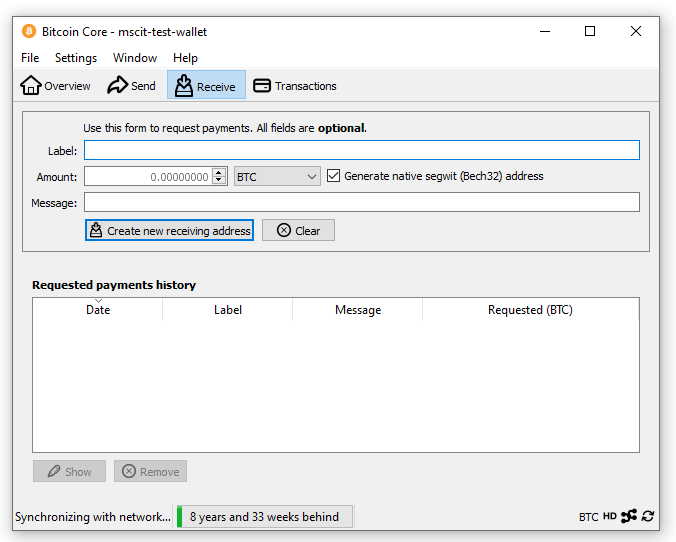
Step 9: Enter Wallet name

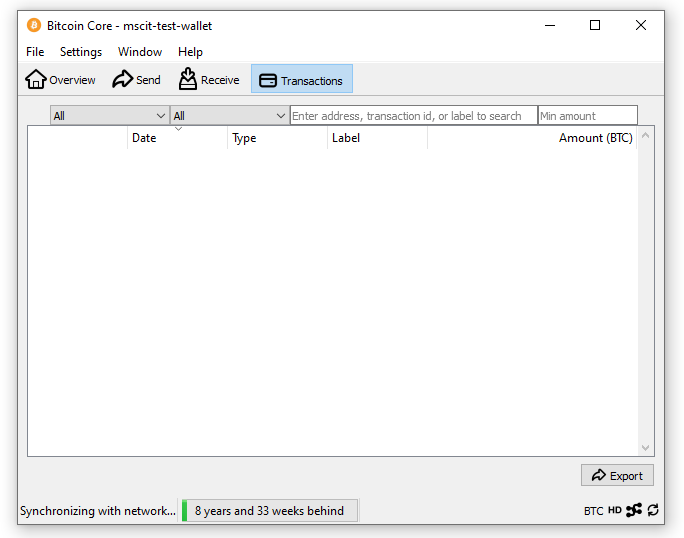


Step 10: Finally Account is setup









**Practical 6**

**A. Demonstrate the use of Bitcoin Core API.**

**Code and Output:**

!pip install bitcoin

from bitcoinlib.wallets import Wallet

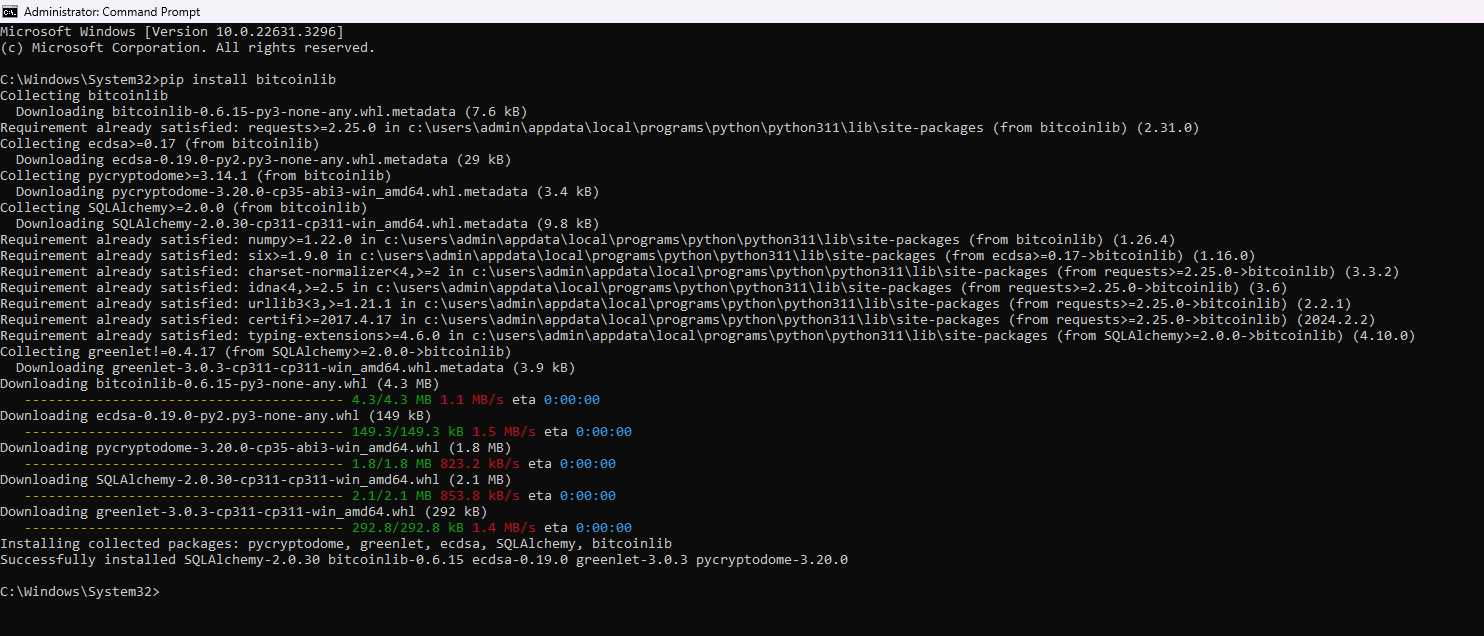
w = Wallet.create('Wallet6')

key1 = w.get\_key()

print('Wallet Address:',key1.address)

w.scan()

print(w.info())





**Practical 7**

**A. Create your own blockchain and demonstrate its use.**

**Steps:**

1. [Install VMWare Workstation](https://drive.google.com/drive/folders/117dXKN249potAMxAJyVux3w1CBvlVbSC?usp=sharing)

2. [Download Linux Virtual Machine](https://drive.google.com/file/d/1CuAxzoMYLHPFrI3hfhCJlmLFMW4YTNc7/view?usp=sharing)

3. Open Linux VM in VMWare

4. If creating new VM. Follow this command before starting step 5. Open new Terminal, on terminal type this command

>sudo add-apt-repository -y ppa:ethereum/ethereum

#if error encountered then run

# sudo apt-get install --reinstall ca-certificates

Install stable version of go-etherium

>sudo apt-get update

>sudo apt-get install ethereum

**A screenshot of a computer

Description automatically generated**

5. Open new Terminal and type this command for creating new directory for storing blockchain data: >**mkdir myblockchain**

**>cd myblockchain**

**>geth account new --datadir data**

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer

Description automatically generated

6. Create genesis.json file

sudo nano genesis.json

{

"config": {

"chainId": 12345,

"homesteadBlock": 0,

"eip150Block": 0,

"eip155Block": 0,

"eip158Block": 0,

"byzantiumBlock": 0,

"constantinopleBlock": 0,

"petersburgBlock": 0,

"istanbulBlock": 0,

"berlinBlock": 0, "ethash": {}

},

"difficulty": "1",

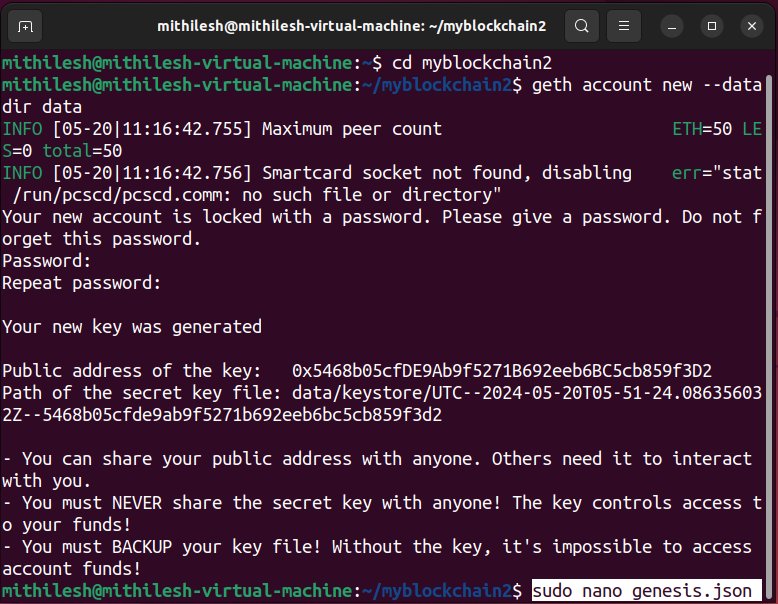
"gasLimit": "8000000", "alloc": {

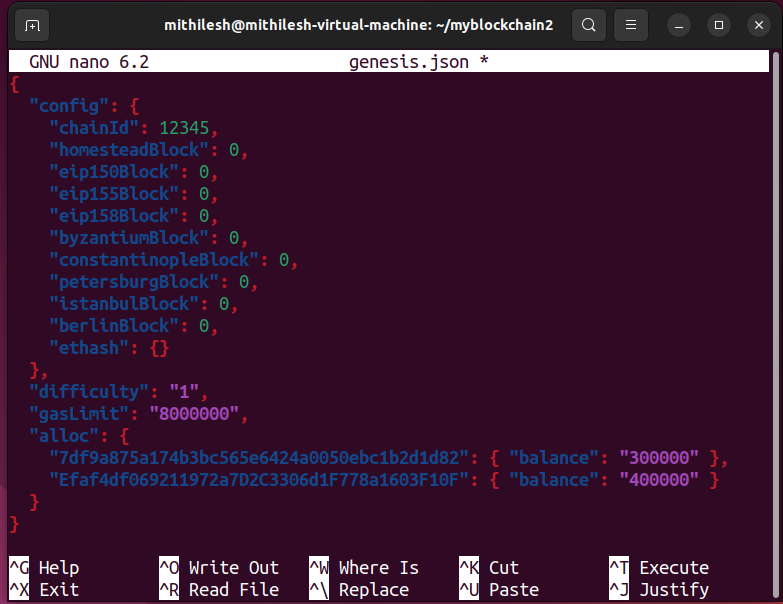
"7df9a875a174b3bc565e6424a0050ebc1b2d1d82": { "balance": "300000" }, "Efaf4df069211972a7D2C3306d1F778a1603F10F": { "balance": "400000" }

}

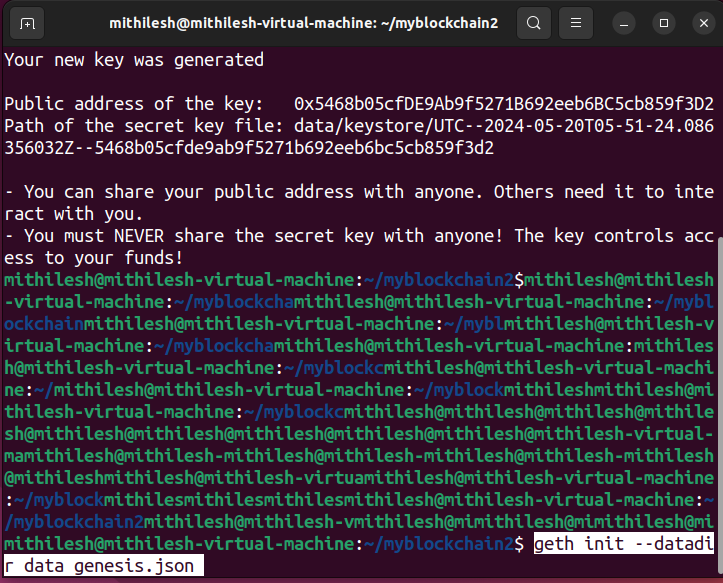
}

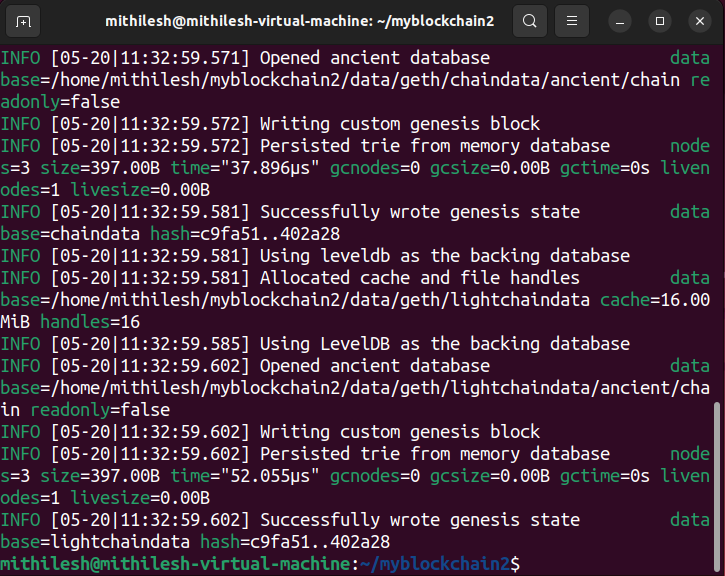
save the file -> “ctrl +o to write” and “{enter} save -> ctrl +x” to exit

****



7. Initialize block “geth init --datadir data genesis.json”





8. Create Network “geth --datadir data --networkid 12345” and **do not close this terminal**

A screenshot of a computer

Description automatically generated

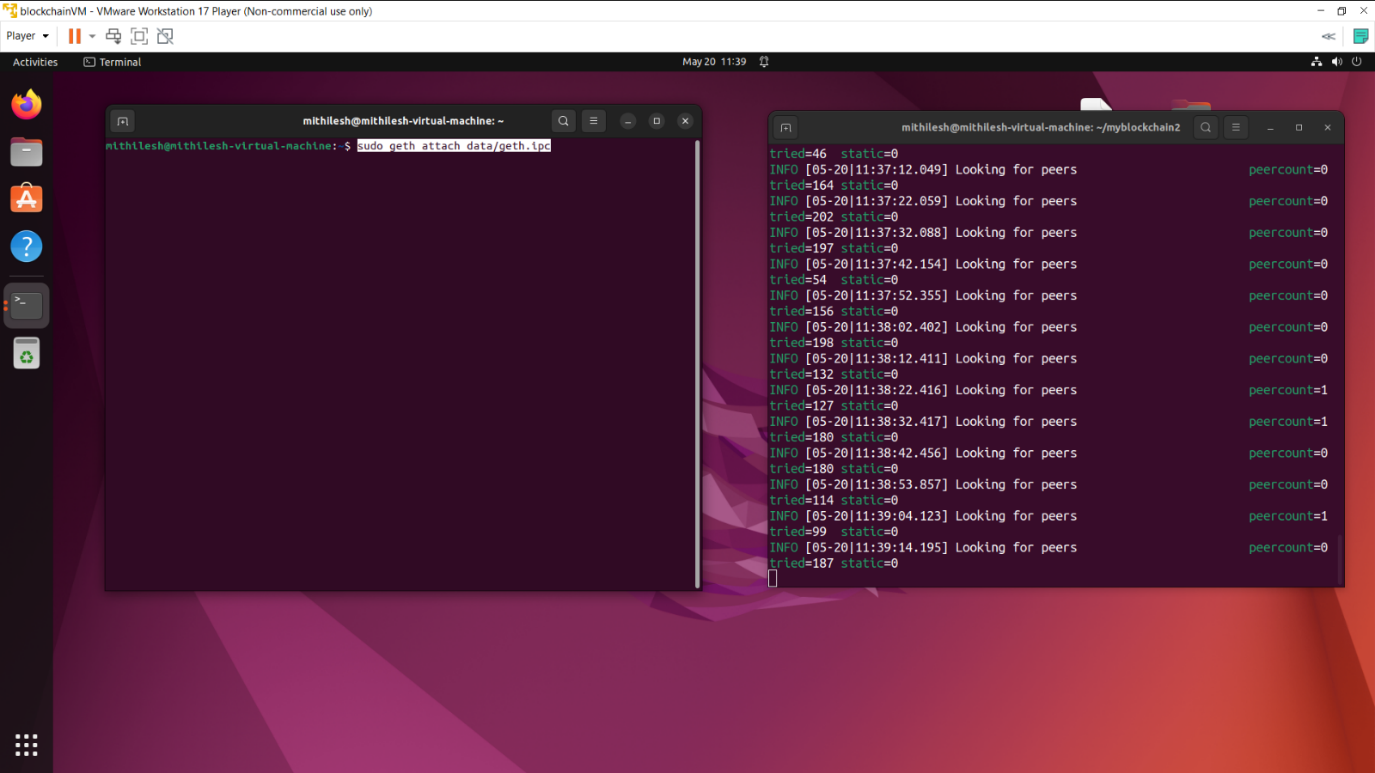
A screenshot of a computer

Description automatically generated

9. Open new terminal

“**sudo geth attach data/geth.ipc eth.getBalance(eth.accounts[0]) miner.setEtherbase(eth.accounts[0])”**

**“miner.start() admin.addPeer(admin.nodeInfo.enode) eth.getBalance(eth.accounts[0])”**



A screenshot of a computer program

Description automatically generated

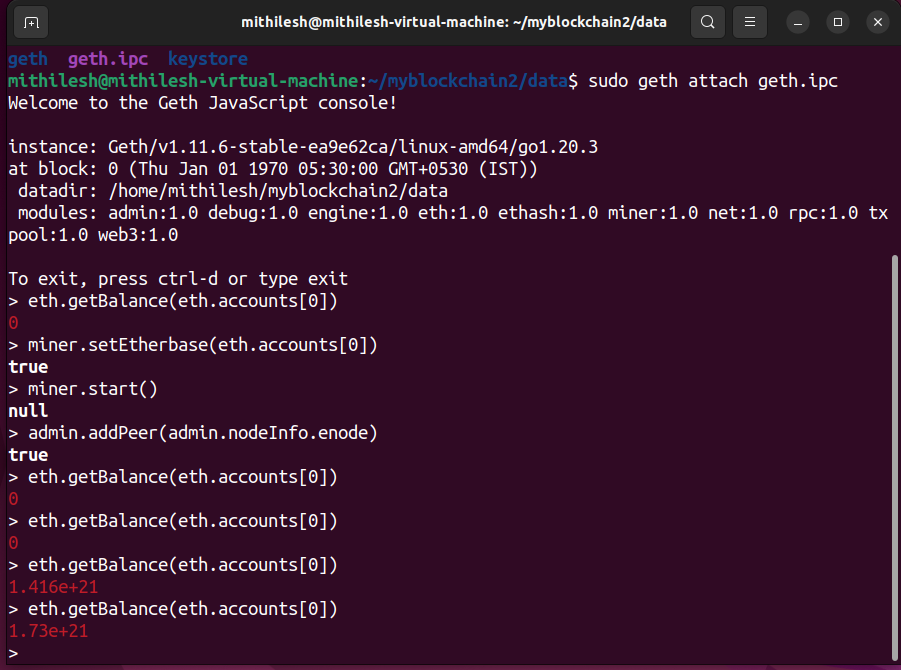
10. **Wait for 10-20 minutes and check balance “eth.getBalance(eth.accounts[0])”**

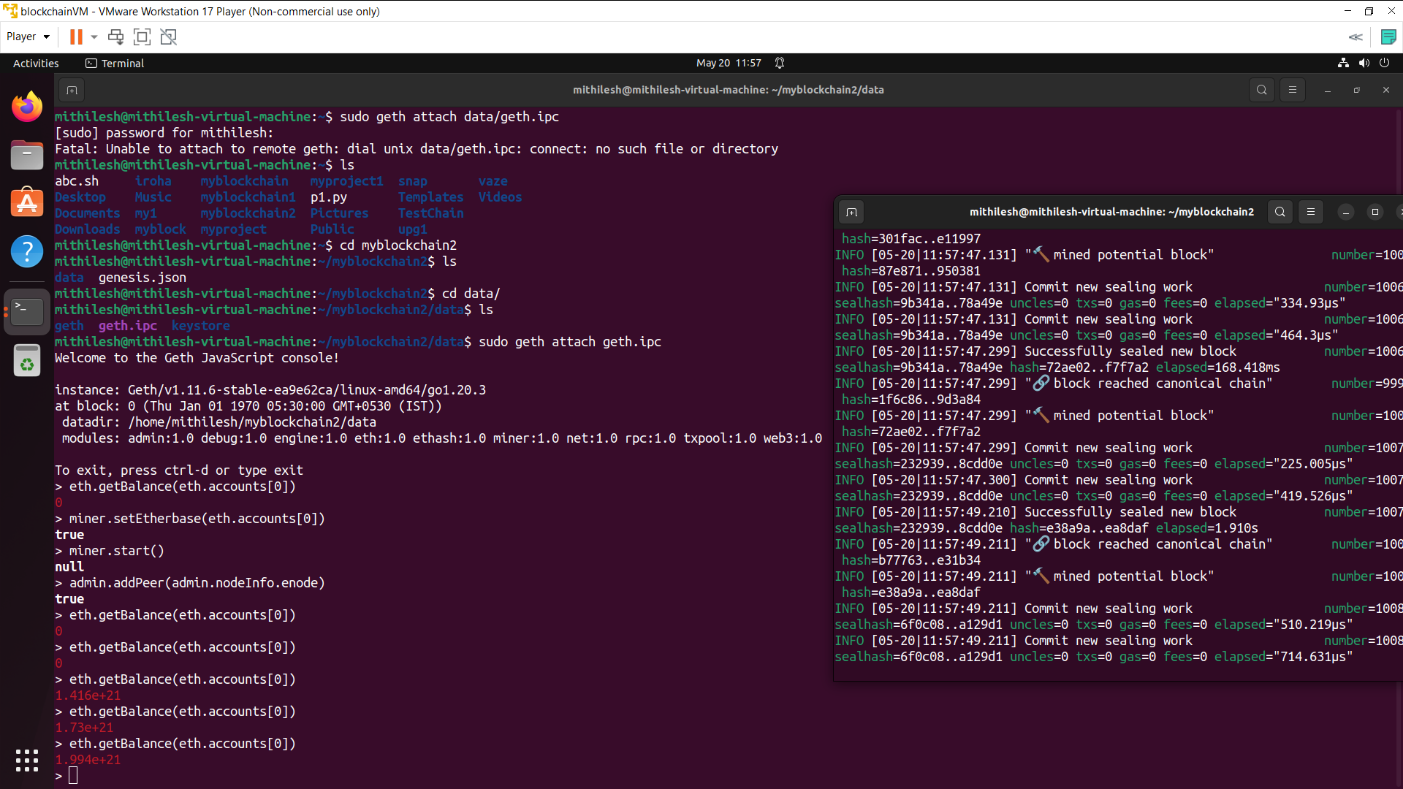
**After balance is updated, you can check current block height**

**>eth.blockNumber**

A screenshot of a computer

Description automatically generated





**Practical 8**

**A. Build D-Apps with angular.**

Step 1:Install the required package –on new terminal 1 type these commands

sudo apt-get -y install curl git vim build-essential sudo apt-get install curl software-properties-common sudo apt install npm

sudo npm install -g web3 sudo apt-get install nodejs sudo apt install python3.9

curl -sL https://deb.nodesource.com/setup\_10.x | sudo bash - sudo npm install --global node-sass@latest

sudo npm install -g truffle@latest sudo npm install -g ganache-cli

export NODE\_OPTIONS=--openssl-legacy-provider

To update npm>sudo npm cache clean -f sudo npm install -g n sudo n latest

Step 2: Start from step 2 if you have VM configured. Create a new directory

mkdir myupg

cd myupg

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer

Description automatically generated

Step 3: Initialize the project folder

truffle init

Step 4: Now create a new contract

nano contracts/HelloWorld.sol

A screenshot of a computer

Description automatically generated

Step 5:Add the following code in HelloWorld.sol pragma solidity ^0.5.0;

contract HelloWorld {

function sayHello() public pure returns(string memory){ return("hello world");

}}

A screenshot of a computer

Description automatically generated

Step 6: create default configuration file: nano migrations/1\_initial\_migration.js

A screenshot of a computer

Description automatically generated

Step 7: Edit this line in the file

const Migrations = artifacts.require("HelloWorld"); module.exports = function (deployer) { deployer.deploy(Migrations,"hello");

};

A screenshot of a computer

Description automatically generated

Step 8: Edit network configuration file

sudo nano truffle-config.js

Remove all line(press CTRL +K) from the file and add the following lines #########################

module.exports = { networks: { development: {

host: "127.0.0.1",

port: 8545, network\_id: "\*",

}

}}

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Step 9: start ganache-cli –Switch/Open to terminal 2 ganache-cli

A screenshot of a computer

Description automatically generated

Step 10: deploy the truffle deploy- On terminal 1 truffle deploy

[Note contract address]

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Step 11: Open truffle console - On terminal 1 truffle console

Step 11: Get reference of contact

contract = await HelloWorld.at('0x2C403EE1b30F56C0c773089c1Eb9DddF1499C969')

A screenshot of a computer screen

Description automatically generated

[Replace '0x2C403EE1b30F56C0c773089c1Eb9DddF1499C969' with your contact address; every time you compile/deploy a new contract address will be generated]

Step 12: Call the function from the contract

a = await contract.sayHello()

a

A screenshot of a computer program

Description automatically generated