**Practical 1**

**Write the following programs for Blockchain in Python**

**Practical 1 a)**

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

**Code:**

#pip install pycryptodome

#1A.- A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.

import Crypto

import binascii

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(self):

# Creating random number for key

random = Crypto.Random.new().read

# Creating new public key and private key

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')

Demo = Client()

print(Demo.identity)

**Output:**

**A computer screen shot

Description automatically generated**

**Practical 1 b)**

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

**Code:**

#1B.- A transaction class to send and receive money and test it.

import Crypto

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto.Hash import SHA

class Client:

def \_\_init\_\_(self):

# Creating random number for key

random = Crypto.Random.new().read

# Creating new public key and private key

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, receiver, value):

self.sender = sender

self.receiver = receiver

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,

'receiver': self.receiver,

'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')

Ninad = Client()

print("-"\*50)

print("Ninad Key")

print(Ninad.identity)

KS = Client()

print("-"\*50)

print("KS Key")

print(KS.identity)

t = Transaction(Ninad, KS.identity, 10.0)

print("-"\*50)

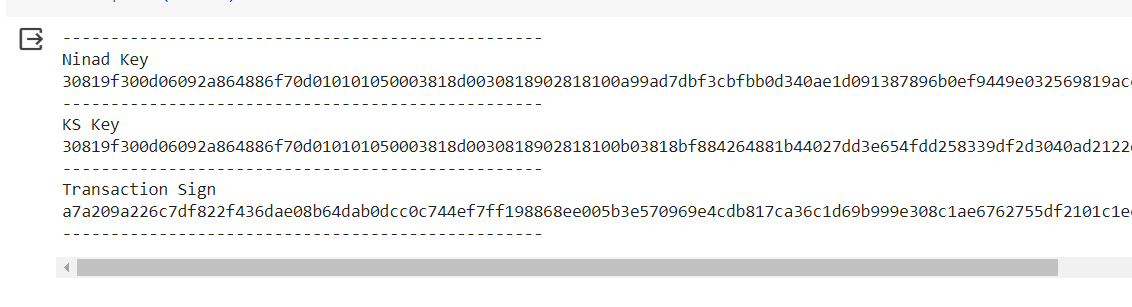
print("Transaction Sign")

signature = t.sign\_transaction()

print(signature)

print("-"\*50)

**Output:**

****

**Practical 1 c)**

**Aim:** Create multiple transactions and display them.

**Code:**

#!pip install pycryptodome

import Crypto

import binascii

from Crypto.PublicKey import RSA

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

import datetime

import collections

import hashlib

from hashlib import sha256

class Client:

def \_\_init\_\_(self):

# Creating random number for key

random = Crypto.Random.new().read

# Creating new public key and private key

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, receiver, value):

self.sender = sender

self.receiver = receiver

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,

"receiver": self.receiver,

"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 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.startwith(prefix):

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

return digest

class Block:

def \_\_init\_\_(self):

self.verified\_transactions = []

self.previous\_block\_hash = ""

self.Nonce = ""

last\_block\_hash = ""

def display\_transaction(transaction):

dict = transaction.to\_dict()

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

print("-----")

print("Receiver: " + dict["receiver"])

print("-----")

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

print("-----")

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

print("-----")

TPCoins = []

def dump\_blockchain(self):

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

for x in range(len(Block.TPCoins)):

block\_temp = Block.TPCoins[x]

print("block #" + str(x))

for transaction in block\_temp.verified\_transactions:

Block.display\_transaction(transaction)

print("-------")

last\_transaction\_index = 0

transactions = []

Ninad = Client()

ks = Client()

vighnesh = Client()

sairaj = Client()

t1 = Transaction(Ninad, ks.identity, 15.0)

t1.sign\_transaction()

transactions.append(t1)

t2 = Transaction(Ninad, vighnesh.identity, 6.0)

t2.sign\_transaction()

transactions.append(t2)

t3 = Transaction(Ninad, sairaj.identity, 16.0)

t3.sign\_transaction()

transactions.append(t3)

t4 = Transaction(vighnesh, Ninad.identity, 8.0)

t4.sign\_transaction()

transactions.append(t4)

t5 = Transaction(vighnesh, ks.identity, 19.0)

t5.sign\_transaction()

transactions.append(t5)

t6 = Transaction(vighnesh, sairaj.identity, 35.0)

t6.sign\_transaction()

transactions.append(t6)

t7 = Transaction(sairaj, vighnesh.identity, 5.0)

t7.sign\_transaction()

transactions.append(t7)

t8 = Transaction(sairaj, Ninad.identity, 12.0)

t8.sign\_transaction()

transactions.append(t8)

t9 = Transaction(sairaj, ks.identity, 25.0)

t9.sign\_transaction()

transactions.append(t9)

t10 = Transaction(Ninad, ks.identity, 1.0)

t10.sign\_transaction()

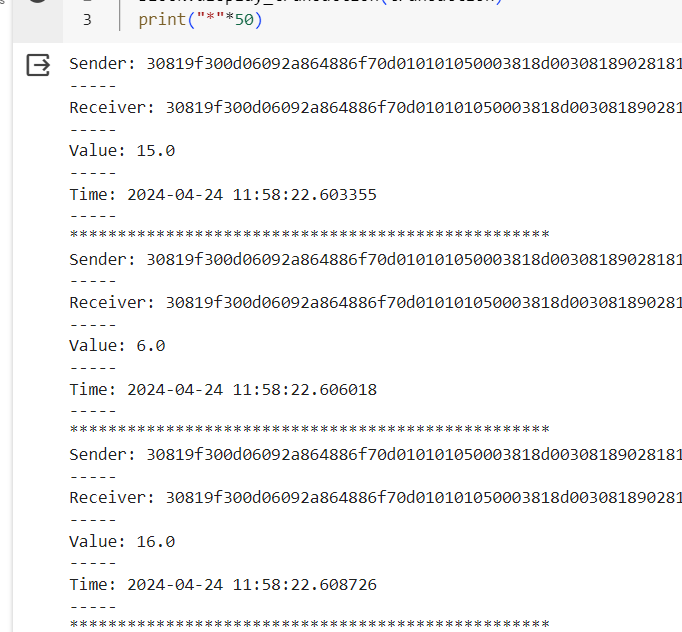
transactions.append(t10)

for transaction in transactions:

display\_transaction(transaction)

print("\*" \* 50)

**Output:**



A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

**Practical 1 d)**

**Aim:** Create a blockchain, a genesis block and execute it.

Code:

# Aim 1D - Create a blockchain, a genesis block and execute it.

#!pip install pycryptodome

import Crypto

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(self):

# Creating random number for key

random = Crypto.Random.new().read

# Creating new public key and private key

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, receiver, value):

self.sender = sender

self.receiver = receiver

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,

"receiver": self.receiver,

"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")

class Block:

def \_\_init\_\_(self):

self.verified\_transactions = []

self.previous\_block\_hash = ""

self.Nonce = ""

last\_block\_hash = ""

def display\_transaction(transaction):

dict = transaction.to\_dict()

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

print("-----")

print("Receiver: " + dict["receiver"]) # Corrected typo

print("-----")

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

print("-----")

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

print("-----")

Ninad = Client()

t0 = Transaction("Genesis", Ninad.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 = []

def dump\_blockchain(self):

print("Number of blocks in 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:

Block.display\_transaction(transaction)

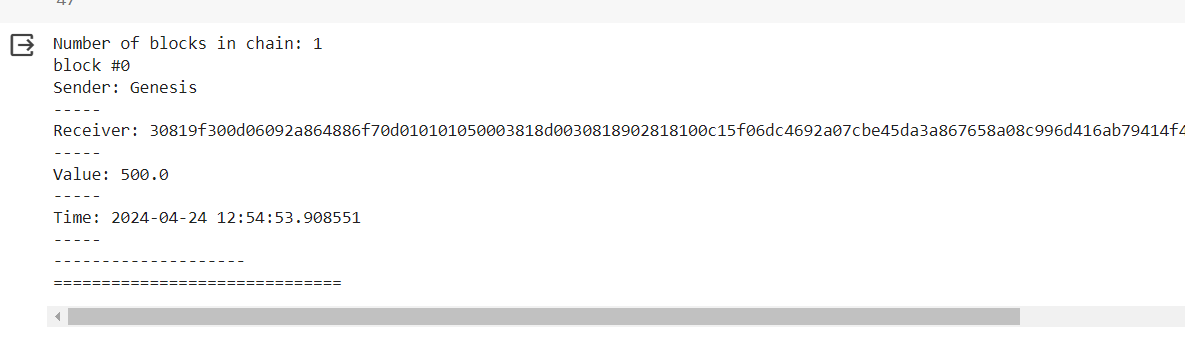
print("-" \* 20)

print("=" \* 30)

TPCoins.append(block0)

dump\_blockchain(TPCoins)

Output:



**Practical 1 e)**

**Aim:** e. Create a mining function and test it and Add blocks to the miner and dump the blockchain.

**Code:**

# -\*- coding: utf-8 -\*-

import collections

import datetime

import binascii

# !pip install pycryptodome

import Crypto

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Hash import SHA

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')

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')

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("after"+str(i)+"iterationsfoundnonce:"+digest)

            return digest

class Block:

    def \_\_init\_\_(self):

        self.verified\_transactions=[]

        self.previous\_block\_hash=""

        self.Nonce=""

def display\_transaction(transaction):

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

last\_block\_hash=""

TPCoins=[]

last\_transaction\_index=0

transactions=[]

Raja=Client()

Rani=Client()

Seema=Client()

Reema=Client()

t1=Transaction(Raja,Rani.identity,15.0)

t1.sign\_transaction()

transactions.append(t1)

t2=Transaction(Raja,Seema.identity,6.0)

t2.sign\_transaction()

transactions.append(t2)

t3=Transaction(Rani,Reema.identity,2.0)

t3.sign\_transaction()

transactions.append(t3)

t4=Transaction(Seema,Rani.identity,4.0)

t4.sign\_transaction()

transactions.append(t4)

t5=Transaction(Reema,Seema.identity,7.0)

t5.sign\_transaction()

transactions.append(t5)

t6=Transaction(Rani,Seema.identity,3.0)

t6.sign\_transaction()

transactions.append(t6)

t7=Transaction(Seema,Raja.identity,8.0)

t7.sign\_transaction()

transactions.append(t7)

t8=Transaction(Seema,Rani.identity,1.0)

t8.sign\_transaction()

transactions.append(t8)

t9=Transaction(Reema,Raja.identity,5.0)

t9.sign\_transaction()

transactions.append(t9)

t10=Transaction(Reema,Rani.identity,3.0)

t10.sign\_transaction()

transactions.append(t10)

#Miner1addsablock

block=Block()

for i in range(3):

    temp\_transaction=transactions[last\_transaction\_index]

    #validatetransaction

    #if valid

    block.verified\_transactions.append(temp\_transaction)

    last\_transaction\_index+=1

block.previous\_block\_hash=last\_block\_hash

block.Nonce=mine(block,2)

digest=hash(block)

TPCoins.append(block)

last\_block\_hash=digest

#Miner2 adds a block

block=Block()

for i in range(3):

    temp\_transaction=transactions[last\_transaction\_index]

    #validate transaction

    #if valid

    block.verified\_transactions.append(temp\_transaction)

    last\_transaction\_index+=1

block.previous\_block\_hash=last\_block\_hash

block.Nonce=mine(block,2)

digest=hash(block)

TPCoins.append(block)

last\_block\_hash=digest

#Miner3 adds a block

block=Block()

for i in range(3):

    temp\_transaction=transactions[last\_transaction\_index]

    #validate transaction

    #if valid

    block.verified\_transactions.append(temp\_transaction)

    last\_transaction\_index+=1

block.previous\_block\_hash=last\_block\_hash

block.Nonce=mine(block,2)

digest=hash(block)

TPCoins.append(block)

last\_block\_hash=digest

dump\_blockchain(TPCoins)

**Output:**

**A close-up of a line

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Practical 2**

**Implement and demonstrate the use of the following in Solidity:**

**Aim: A)** Variable and Operators.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract PrimitiveDataTypes {

uint8 a = 20;

uint256 b = 35;

int c = 10;

int8 d = 3;

bool flag = true;

address public addr;

constructor() {

addr = msg.sender;

}

uint public addition = a + b;

int public subtraction = c - d;

int public multiply = d \* c;

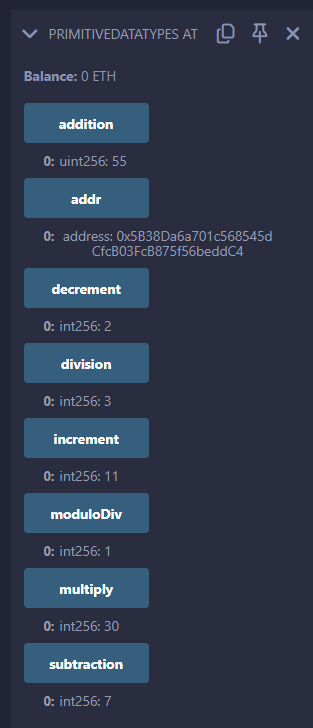
int public division = c / d;

int public moduloDiv = c % d;

int public increment = ++c;

int public decrement

**Output:**

****

**Aim: B)Loops**.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Loop {

function summation(uint256 n) public pure returns (uint256) {

uint256 sum = 0;

for (uint256 i = 1; i <= n; i++) {

sum += i;

}

return sum;

}

function sumWhile(uint256 n) public pure returns (uint256) {

uint256 sum = 0;

uint256 i = 1;

while (i <= n) {

sum += i;

i++;

}

return sum;

}

function sumDoWhile(uint256 n) public pure returns (uint256) {

uint256 sum = 0;

uint256 i = 1;

do {

sum += i;

i++;

} while (i <= n);

return sum;

}

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Aim: C) Decision Making**.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract decision{

    function even(uint n) public pure returns(bool){

        if(n%2==0){

            return true;

        }

        else{

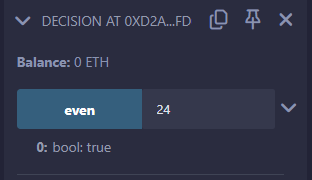
            return false;

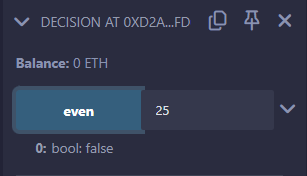
        }

    }

}

**Output:**

****

****

**Aim: D) Arrays**.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Arrays {

    // Declaring an array

    uint[] public array1 = [1, 2, 3, 4, 5];

    function fetch(uint index) public view returns (uint) {

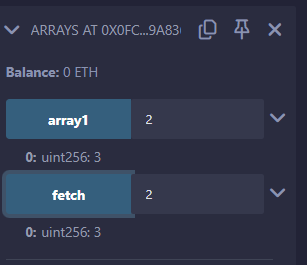
        require(index < array1.length, "Index out of bounds");

        return array1[index];

    }

}

**Output:**

****

**Aim: E) Enums**.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Enums {

enum week\_days {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}

week\_days choice;

function set\_value() public {

choice = week\_days.Friday;

}

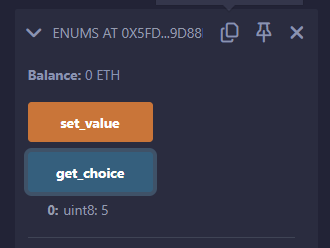
function get\_choice() public view returns (week\_days) {

return choice;

}

}

**Output:**

****

**Aim: F) Structs**.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Structs {

struct Book {

string name;

string writer;

uint price;

bool available;

}

Book book1;

Book book2 = Book("Harry Potter", "J.K.Rowling", 300, true);

function set\_book\_detail() public {

book1 = Book("Introducing Ethereum and Solidity", "Chris Dannen", 250, true);

}

function book1\_info() public view returns (string memory, string memory, uint, bool) {

return(book2.name, book2.writer, book2.price, book2.available);

}

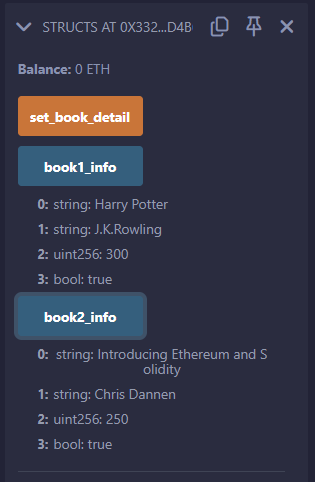
function book2\_info() public view returns (string memory, string memory, uint, bool) {

return (book1.name, book1.writer, book1.price, book1.available);

}

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Aim: G) Mappings**.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract maps {

mapping (uint => string) public roll\_no;

function set(uint keys, string memory value) public {

roll\_no[keys] = value;

}

}

**Output:**

A screenshot of a computer

Description automatically generated

**Aim: H)** Conversions, Ether Units, Special Variables.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract conversion {

    uint   a = 5;    // Default unsigned integer type

    uint8  b = 10;   // 8-bit unsigned integer

    uint16 c = 15;   // 16-bit unsigned integer

    function convert() public view returns (uint) {

        uint result = a + uint(b) + uint(c); // Convert b and c to uint and add them to a

        return result; // Return the result

    }

}

**Output**:

**A screenshot of a computer

Description automatically generated**

**Aim: I)** Strings.

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract StringExample {

string public greeting = "Hello, ";

function concatenate(string memory \_name) public view returns (string memory) {

return string(abi.encodePacked(greeting, \_name));

}

function compareStrings(string memory \_a, string memory \_b) public pure returns (bool) {

return keccak256(abi.encodePacked(\_a)) == keccak256(abi.encodePacked(\_b));

}

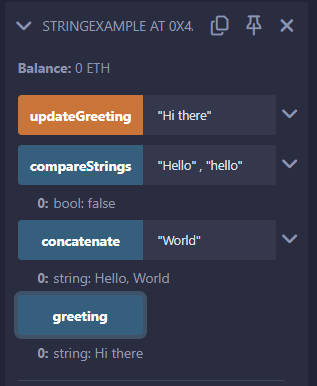
function updateGreeting(string memory \_newGreeting) public {

greeting = \_newGreeting;

}

}

**Output:**

****

**Practical 3**

**Implement and demonstrate the use of the following in Solidity:**

**Aim: A) Function**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Addition {

    int public input1;

    int public input2;

    function setInputs(int \_input1, int \_input2) public {

        input1 = \_input1;

        input2 = \_input2;

    }

    function additions() public view returns(int) {

        return input1 + input2;

    }

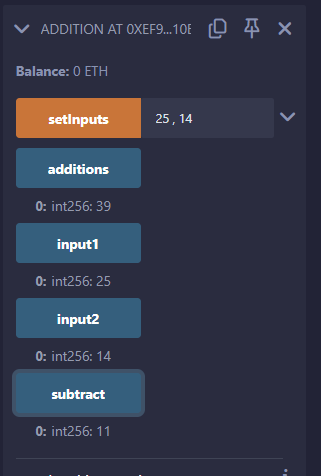
    function subtract() public view returns(int) {

        return input1 - input2;

    }

}

**Output:**

****

**Aim: B) Fallback Function**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract fallbackfn {

    // Event to log details when fallback or receive function is called

    event Log(string func, address sender, uint value, bytes data);

    // Fallback function to handle calls to the contract with data or no matching function

    fallback() external payable {

        emit Log("fallback", msg.sender, msg.value, msg.data); // Emit log with details

    }

    // Receive function to handle plain ether transfers

    receive() external payable {

        emit Log("receive", msg.sender, msg.value, ""); // Emit log with details (msg.data is empty)

        //msg.data is empty hence no need to specify it and mark it as empty string

    }

}

**Output:**

**A computer screen shot of a computer code

Description automatically generated**

**Aim: C) c. Mathematical functions.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract MathOperations {

    // addMod computes (x + y) % k

    // mulMod computes (x \* y) % k

    // Function to compute modular addition and multiplication

    // @return addModResult: Result of (x + y) % k

    // @return mulModResult: Result of (x \* y) % k

    function computeMod() public pure returns (uint addModResult, uint mulModResult) {

        uint x = 3;

        uint y = 2;

        uint k = 6;

        addModResult = addmod(x, y, k); // Compute (x + y) % k

        mulModResult = mulmod(x, y, k); // Compute (x \* y) % k

    }

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Aim: D) d. Cryptographic functions.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Crypto {

    function hash(string memory \_text,uint \_num,address \_addr) public pure returns (bytes32) {

            return keccak256(abi.encodePacked(\_text, \_num, \_addr));

            }

    function collision(string memory \_text, string memory \_anotherText)public pure returns (bytes32){

                return keccak256(abi.encodePacked(\_text, \_anotherText));

            }

}

contract GuessTheWord {

    bytes32 public answer = 0x5f38993891425af42a69bd3cbabdc916f093d4f444455134d4371f4ddd17bd08;

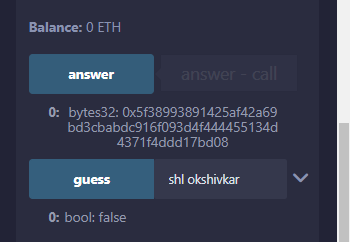
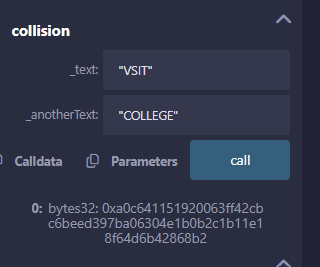
    function guess(string memory \_word) public view returns (bool) {

     return keccak256(abi.encodePacked(\_word)) == answer;

    }

}

**Output:**

****

**Aim: E) e. Function Modifiers.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;

contract FunctionModifier {

    address public owner;

    uint256 public x = 100;

    bool public locked;

    constructor() {

        // Set the transaction sender as the owner of the contract.

        owner = msg.sender;

    }

    modifier onlyOwner() {

        require(msg.sender == owner, "Not owner");

        \_;

    }

    modifier validAddress(address \_addr) {

        require(\_addr != address(0), "Not valid address");

        \_;

    }

    function changeOwner(address \_newOwner)

        public

        onlyOwner

        validAddress(\_newOwner)

    {

        owner = \_newOwner;

    }

    modifier noReentrancy() {

        require(!locked, "No reentrancy");

        locked = true;

        \_;

        locked = false;

    }

    function decrement(uint256 i) public noReentrancy {

        x -= i;

        if (i > 1) {

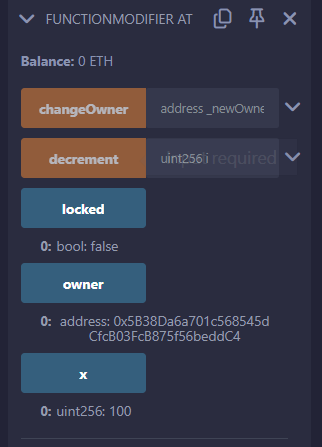
            decrement(i - 1);

        }

    }

}

**Output:**

****

**A screenshot of a computer

Description automatically generated**

**Aim: F) f. View and Pure Functions.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.3;

contract ViewAndPure {

    uint public x = 1;

    // Promise not to modify the state.

    function addToX(uint y) public view returns (uint) {

        return x + y;

    }

    // Promise not to modify or read from the state.

    function add(uint i, uint j) public pure returns (uint) {

        return i + j;

    }

}

**Output:**

****

**Aim: G)** **g. Function Overloading.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract FunctionOverloading {

    // Function with one parameter

    function sum(uint a) public pure returns (uint) {

        return a + 10;

    }

    // Overloaded function with two parameters

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

        return a + b;

    }

    // Overloaded function with three parameters

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

        return a + b + c;

    }

    // Examples of calling overloaded functions

    function exampleUsage() public pure returns (uint, uint, uint) {

        uint result1 = sum(5);            // Calls the first sum function

        uint result2 = sum(5, 10);        // Calls the second sum function

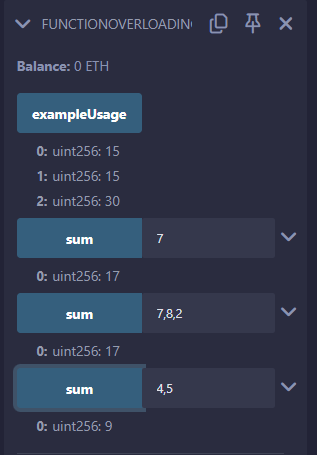
        uint result3 = sum(5, 10, 15);    // Calls the third sum function

        return (result1, result2, result3);

    }

}

**Output:**

****

**Practical 4**

**Implement and demonstrate the use of the following in Solidity:**

**Aim: A) a. Withdrawal Pattern.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;

contract withdrawalPattern{

    address public richest;

    uint public mostSent;

    mapping (address=>uint) pendingWithdrawals;

    error NotEnoughEther();

    constructor() payable{

        richest = msg.sender;

        mostSent = msg.value;

    }

    function becomeRichest() public payable{

        if (msg.value <= mostSent) revert NotEnoughEther();

        pendingWithdrawals[richest] += msg.value;

        richest = msg.sender;

        mostSent = msg.value;

    }

    function withdraw() public {

        uint amount = pendingWithdrawals[msg.sender];

        pendingWithdrawals[msg.sender] = 0;

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

    }

}

**Output:**

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Aim: B) b. Restricted Access.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract AccessRestriction {

    address public owner = msg.sender;

    uint public creationTime = block.timestamp;

    error Unauthorized();

    error TooEarly();

    error NotEnoughEther();

    modifier onlyBy(address account){

        if (msg.sender != account)

        revert Unauthorized();

        \_;

    }

    modifier costs(uint amount) {

        if (msg.value < amount)

            revert NotEnoughEther();

            \_;

        if (msg.value > amount)

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

    }

    modifier onlyAfter(uint time) {

        if (block.timestamp < time)

            revert TooEarly();

            \_;

    }

    function changeOwner(address newOwner)public onlyBy(owner){

        owner = newOwner;

    }

    function disown()public onlyBy(owner) onlyAfter(creationTime + 6 weeks){

        delete owner;

    }

    function forceOwnerChange(address newOwner)public payable costs(200 ether){

        owner = newOwner;

        // just some example condition

        if (uint160(owner) & 0 == 1)

            return;

    }

}

**Output:**

**A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated**

**Practical 5**

**Implement and demonstrate the use of the following in Solidity:**

**Aim: A) a. Contracts and Inheritance.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract C{

    uint private data;

    uint public info;

    constructor()  {

        info = 10;

        }

        function increment(uint a) private pure returns(uint){

            return a + 1;

        }

        function updateData(uint a) public {

            data = a;

        }

        function getData() public view returns(uint) {

            return data;

        }

        function compute(uint a, uint b) internal pure returns (uint) {

            return a + b;

        }

}

contract D {

    function readData() public returns(uint) {

        C c = new C();

        c.updateData(7);

        return c.getData();

    }

}

contract E is C {

    uint private result;

    C private c;

    constructor()  {

        c = new C();

    }

    function getComputedResult() public {

        result = compute(3, 6);

    }

    function getResult() public view returns(uint) {

        return result;

    }

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Aim: B) b. Constructors**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract constructors{

    string str;

    uint amount;

    constructor(){

        str  = "Shlok is learning Solidity";

        amount = 10;

    }

    function const()public view returns(string memory,uint){

        return (str,amount);

    }

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Aim: C) c. Abstract Contracts.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

abstract contract Main {

    // Define an abstract function that can be overridden

    function add(uint a, uint b) public virtual pure returns (uint);

}

contract Adder is Main {

    // Override the add function from the Main contract

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

        return a + b;

    }

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**Aim: D) d. Interfaces**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

interface adder{

    function add(uint a, uint b)external pure returns(uint);

}

contract adderContract is adder{

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

        return a+b;

    }

}

**Output:**

**A screenshot of a phone

Description automatically generated**

**Practical 6**

**Implement and demonstrate the use of the following in Solidity:**

**Aim: a. Libraries.**

**Code:**

pragma solidity ^0.8.17;

library Search {

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

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

         if (self[i] == value) {

            return i;

         }

      }

      return type(uint).max;

   }

}

contract Test {

   uint[] data;

   constructor() {

      data.push(1);

      data.push(2);

      data.push(3);

      data.push(4);

      data.push(5);

   }

   function isValuePresent() external view returns (uint) {

      uint value = 4;

      uint index = Search.indexOf(data, value);

      return index;

   }

}

library MathLibrary {

   function square(uint num) internal pure returns (uint) {

      return num \* num;

   }

}

contract SquareContract {

   using MathLibrary for uint;

   function calculateSquare(uint num) external pure returns (uint) {

      return num.square();

   }

}

**Output:**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Aim: b. Assembly**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

library Sum {

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

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

         assembly {

            // Load the value from memory at the current index

            let value := mload(add(add(\_data, 0x20), mul(i, 0x20)))

            sum := add(sum, value)

         }

      }

      return sum;

   }

}

contract Test {

   uint[] data;

   constructor() {

      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:**

**A screenshot of a computer

Description automatically generated**

**Aim: c. Error handling.**

**Code:**

pragma solidity ^0.8.17;

contract ErrorHandlingExample {

    constructor() payable {

        // Allow the contract to receive Ether during deployment

    }

    function divide(uint256 numerator, uint256 denominator) external pure returns (uint256) {

        require(denominator != 0, "Division by zero is not allowed");

        return numerator / denominator;

    }

    function withdraw(uint256 amount) external {

        require(amount <= address(this).balance, "Insufficient balance");

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

    }

    function assertExample() external pure {

        uint256 x = 5;

        uint256 y = 10;

        assert(x < y);

    }

    function tryCatchExample() external view returns (bool) {

        try this.divide(10, 5) returns (uint256 result) {

            return true;

        } catch {

            return false;

        }

    }

}

**Output:**

A screenshot of a computer

Description automatically generated

**Aim: d. Events.**

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract EventExample {

    event Deposit(address indexed from, uint256 amount);

    event Withdraw(address indexed to, uint256 amount);

    mapping(address => uint256) public balances;

    function deposit() public payable {

        require(msg.value > 0, "Must deposit more than 0 ether");

        balances[msg.sender] += msg.value;

        emit Deposit(msg.sender, msg.value);

    }

    function withdraw(uint256 amount) public {

        require(balances[msg.sender] >= amount, "Insufficient balance");

        balances[msg.sender] -= amount;

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

        emit Withdraw(msg.sender, amount);

    }

}

**Output:**

A screenshot of a computer

Description automatically generated

**Practical 7**

**Install Hyperledger fabric:**

**Aim: Install hyperledger fabric**

**Commands and Output:**

A screenshot of a computer program

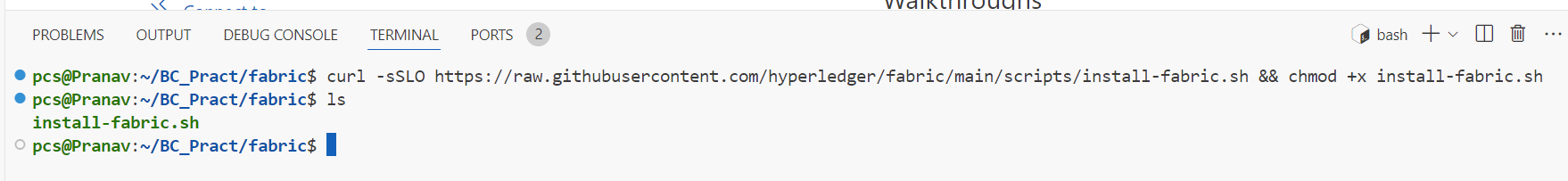
Description automatically generated

A screen shot of a computer code

Description automatically generated

**Download fabric samples**

curl -sSLO https://raw.githubusercontent.com/hyperledger/fabric/main/scripts/install-fabric.sh && chmod +x install-fabric.sh



**Pull the docker containers**

./install-fabric.sh

A screenshot of a computer

Description automatically generated

**Navigate to test network directory**

ls

cd fabric-samples

ls

A screenshot of a computer

Description automatically generated

cd test-network

ls

A close-up of a computer screen

Description automatically generated

**Remove any containers or artifacts**

./network.sh down

A screenshot of a computer

Description automatically generated

**Up the network**

./network.sh up

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Create a channel**

./network.sh createChannel

A screenshot of a computer

Description automatically generated

**Deploy chaincode on peers and channel**

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-javascript -ccl javascript

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

***Interacting with the network***

**Set the path for peer binary and config for core.yaml**

export PATH=${PWD}/../bin:$PATH

export FABRIC\_CFG\_PATH=$PWD/../config/

**Set the environment variables to operate Peer as Org1**

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

A screenshot of a computer

Description automatically generated

**Command to initialize the ledger with assets**

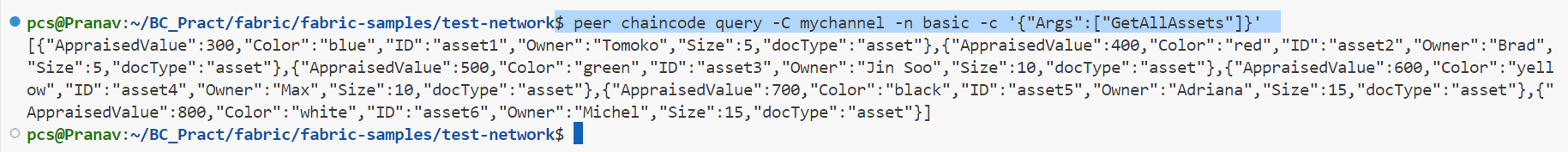
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'

A screenshot of a computer

Description automatically generated

**Query the ledger**

peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'



**Transfer the asset**

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"TransferAsset","Args":["asset6","Christopher"]}'

A close-up of a text

Description automatically generated

**Lets query the ledger from Org2 peer**

**Set the environment variables to operate Peer as Org2**

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org2MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

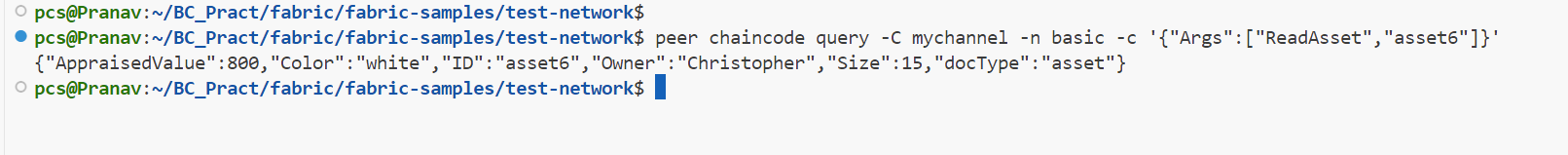
export CORE\_PEER\_ADDRESS=localhost:9051

A close-up of a text

Description automatically generated

**Query the ledger**

peer chaincode query -C mychannel -n basic -c '{"Args":["ReadAsset","asset6"]}'



**Bring the network down**

./network.sh down

**Practical 8**

**Demonstrate the running of the blockchain node (create node using solidity and run).**

**To check if the prerequisites (Node.js, npm, and Truffle) are installed, you can run the following commands:**

**Step 1: Prerequisites**

**Install Node.js**

<https://nodejs.org/en/download/prebuilt-installer>

**Execute the following Commands:**

npm install -g truffle

npm install -g ganache-cli

1. **Check Node.js and npm installation:**

node -v

npm -v

1. **Check Truffle installation:**

truffle version

A screenshot of a computer program

Description automatically generated

1. **Install Ganache**

<https://archive.trufflesuite.com/ganache/>

A screenshot of a computer

Description automatically generated

1. **Create a new Workspace (BC\_Pract)**

A screenshot of a computer

Description automatically generated

**Step 2: Initialize a Truffle Project**

1. **Create a new directory for your project:**

mkdir myProj

cd myProj

1. **Initialize the Truffle project:**

truffle init

**Step 3: Create a Solidity Smart Contract**

1. **Navigate to the Contracts directory(myProj/contracts):**

**SimpleStorage.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SimpleStorage {

uint256 public storedData;

function set(uint256 x) public {

storedData = x;

}

function get() public view returns (uint256) {

return storedData;

}

}

1. **Compile the Smart Contract**

Command: truffle compile

**C:\Users\prana\Desktop\BC\_Pract\Pract\_8\myProj>truffle compile**

**Step 4: Configure Truffle to Use Ganache**

**Open the truffle-config.js file and configure the development network to use Ganache. Update the networks section:**

module.exports = {

networks: {

development: {

host: "127.0.0.1",

port: 7545, // Match the port Ganache is using

network\_id: "\*" // Match any network id

}

},

compilers: {

solc: {

version: "0.8.0" // Specify the Solidity compiler version

}

}

};

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

**Step 5: Migrate the Smart Contract to Ganache**

1. **Start Ganache (open the Ganache application and start a new workspace(BC\_Pract)).**
2. **Create a migration script in the migrations directory   
   (e.g., deploy\_contracts.js):**

**Pract\_8\myProj\migrations\2\_deploy\_contracts.js**

const SimpleStorage = artifacts.require("SimpleStorage");

module.exports = function (deployer) {

deployer.deploy(SimpleStorage);

};

A screen shot of a computer

Description automatically generated

1. **Run the migration:**

**Command: truffle migrate**

C:\Users\prana\Desktop\BC\_Pract\Pract\_8\myProj>truffle migrate

A screenshot of a computer program

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

**Step 6: Interact with the Deployed Contract**

1. **Open the new command prompt:**

**Command**: truffle console

C:\Users\prana\Desktop\BC\_Pract\Pract\_8\myProj>truffle console

1. **Interact with the deployed contract:**

Execute the following commands one-by-one

let instance = await SimpleStorage.deployed()

await instance.set(42)

let value = await instance.get()

value.toString() // Output should be '42'

A screenshot of a computer program

Description automatically generated A screen shot of a computer code

Description automatically generated

**Practical 9**

**Demonstrate the use of Bitcoin API:**

**Aim:**

import requests

# Task 1: Get information regarding the current block

def get\_current\_block\_info():

    response = requests.get("https://blockchain.info/latestblock")

    block\_info = response.json()

    print("Current block information:")

    print("Block height:", block\_info['height'])

    print("Block hash:", block\_info['hash'])

    print("Block index:", block\_info['block\_index'])

    print("Timestamp:", block\_info['time'])

# Task 3: Get balance of an address

def get\_address\_balance(address):

    response = requests.get(f"https://blockchain.info/q/addressbalance/{address}")

    balance = float(response.text) / 10\*\*8

    print("Balance of address", address, ":", balance, "BTC")

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    # Task 1: Get information regarding the current block

    get\_current\_block\_info()

    # Task 3: Get balance of an address

    address = "3Dh2ft6UsqjbTNzs5zrp7uK17Gqg1Pg5u5"

    get\_address\_balance(address)

**Output:**

**A screen shot of a computer program

Description automatically generated**