Practical 1

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

```
pip install –upgrade pip
pip install pycryptodomex or Pip install pycryptodome
pip install RSA
```

```
from Crypto.PublicKey import RSA
key = RSA.generate(2048)
p_key = key.public_key().export_key('PEM')
priv_key = key.export_key('PEM')
print("sayali\n")
print(p_key)
print(priv_key)
```

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

```
class Bank:
    def __init__(self) -> None:
        self.balance = 0
    def deposit(self):
        amount = float(input("enter amount to be deposited"))
        self.balance = self.balance + amount
        print(self.balance)
    def withdraw(self):
        amount = float(input("enter amount to be withdraw"))
        if(self.balance >= amount):
            self.balance = self.balance - amount
            print(self.balance)
        else:
            print("no balance")
    def enquiry(self):
        print(self.balance)

acc= Bank()
acc.deposit()
acc.withdraw()
acc.enquiry()
```

C) Create multiple transactions and display them.

```
blockchain =[]
def get_last_value():
    return blockchain[-1]
def add_value(sender,receipent,amount=1.0):
    transaction = {'sender':sender,
                   'receipent':receipent,
                   'amount':amount}
    blockchain.append(transaction)
def get_transaction_value():
    tx_sender = input('Enter the sender: ')
    tx_recipient = input('Enter the recipient pf the transaction: ')
    tx_amount = float(input('Enter your transaction amount: '))
    return tx_sender, tx_recipient, tx_amount

def print_block():
    for block in blockchain:
        print("Sayali \n")
        print("here is your block")
        print(block)
again = True
while again == True:
    tx = get_transaction_value()
    s, r, a = tx
    add_value(s, r, a)
    print(blockchain)
    more = input("add more block (Y/N)? ")
    if more.lower() == 'y':
        again = True
    else:
        again = False
```

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

Code :

```
import hashlib
import time

class Block:
        def__init_(self, index, previous_hash, transactions, timestamp):
        self.index = index
        self.previous_hash = previous_hash
        self.transactions = transactions
        self.timestamp = timestamp
        self.hash = self.calculate_hash()

        def calculate_hash(self):
        data_string = str(self.index) + self.previous_hash + str(self.transactions) +
str(self.timestamp)
        return hashlib.sha256(data_string.encode()).hexdigest()

class Blockchain:
        def__init_(self):
        self.chain = [self.create_genesis_block()]

        def create_genesis_block(self):
        return Block(0, "0", [], time.time())

        def add_block(self, new_block):
        new_block.previous_hash = self.chain[-1].hash
        new_block.hash = new_block.calculate_hash()
        self.chain.append(new_block)
```

```
# Create a blockchain
my_blockchain = Blockchain()

# Create a genesis block
genesis_block = my_blockchain.chain[0]
print("Genesis Block:")
print("Index:", genesis_block.index)
print("Previous Hash:", genesis_block.previous_hash)
print("Transactions:", genesis_block.transactions)

print("Timestamp:", genesis_block.timestamp)
print("Hash:", genesis_block.hash)
print()

# Create a new block
new_block = Block(1, genesis_block.hash, ["Transaction 1", "Transaction 2"], time.time())
my_blockchain.add_block(new_block)

# Print the new block
print("New Block:")
print("Index:", new_block.index)
print("Previous Hash:", new_block.previous_hash)
print("Transactions:", new_block.transactions)
print("Timestamp:", new_block.timestamp)
print("Hash:", new_block.hash)
```

E. Create a mining function and test it.

Code :

```
import hashlib


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


def mine(message, difficulty=1):
    assert difficulty >= 1
    prefix = "1" * difficulty
    for i in range(1000):
    digest = sha256(str(hash(message)) + str(i))
    if digest.startswith(prefix):
    print(f"after {str(i)} iterations found nonce: {digest}") # return print(digest)

mine("test message", 2)
```

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

Code :

```
import datetime
import hashlib

class Block:
      def      init_(self, data, previous_hash):
        self.timestamp = datetime.datetime.now(datetime.timezone.utc)
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.calc_hash()

        def calc_hash(self):
        sha = hashlib.sha256()
        hash_str = self.data.encode("utf-8")
        sha.update(hash_str)
        return sha.hexdigest()


# Instantiate the class

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

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

# Dumping the blockchain

for block in blockchain:
        print(f"Timestamp: {block.timestamp}\nData: {block.data}\nPrevious
Hash:{block.previous_hash}\nHash: {block.hash}\n")
```

Practical 2 :

Aim : Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.(MetaMask & Remix)

Steps :

Step 1-> Install MetaMask extension for chrome from Chrome Web Store
Step 2-> Click on Metamask Extension in Extensions. Below page will open in a new tab. Click on Create a New Wallet. Click on I agree.
Step 3-> Create a password. This password can be used only on the device it was created on. Create a Strong password and click on Create a new Wallet Button
Step 4-> Click on Secure my wallet button, following window will appear
Step 5-> Click on Reveal Secret Recovery Phrase button and save the words in the same sequence
Step 6-> Enter the respective words in the empty positions and click Confirm.
Step 7-> Click Got it!
Step 8 -> Click on Next
Step 9-> Following will be the Dashboard

Step 10-> Click on Ethereum Mainnet button. Next click on Show/hide test Networks.
Step 11-> Check if tesnets are shown by clicking on Etherum Mainnet button. Click on Sepolia test network.
Step 12-> Go to https://sepoliafaucet.com/ and Click on Alchemy Login button.
Step 13-> Login to a gmail account in another browser tab and click on Sign in with Google
Step 14-> Now go to MetaMask and copy the account address.
Step 15-> Paste the address and click on Send Me ETH.
Step 16-> Your ETH transfer is succesfull. You should see a similar animation.
Step 17-> Check your MetaMask account for Sepolia test network. 0.5 ETH will be Added.

PRACTICAL-3 IMPLEMENT AND DEMONSTRATE THE USE OF THE

FOLLOWING IN SOLIDITY

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

A. Variable

Code :

```
pragma solidity ^0.5.0;
contract SolidityTest {
   uint storedData; // State variable
   constructor() public {
         storedData = 10;
   }
   function getResult() public pure returns(uint){
         uint a = 1; // local variable
         uint b = 2;
         uint result = a + b;
         return result; //access the local variable
   }}
```

Strings:

Code :

```
pragma solidity ^0.5.0;

contract LearningStrings { string text;

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

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

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

Operatoprs:

```
pragma solidity ^0.5.0;

contract SolidityTest {
uint16 public a = 20;
uint16 public b = 10;
uint256 public sum = a + b;
uint256 public diff = a - b;
uint256 public mul = a * b;
uint256 public div = a / b;
uint256 public mod = a % b;
uint256 public dec = --b;
uint256 public inc = ++a;
}
```

4. Array

Code:

```
pragma solidity ^0.5.0; contract arraydemo
{
uint[6] arr2=[10,20,30];
function dispstaticarray() public view returns(uint[6] memory)
{
        return arr2;
}
uint x=5;
uint [] arr1;

function arrayDemo() public

{
```

```
            while(x >0)
            {

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

}

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

{

return arr1;

}

}
```

5. Decision Making

If Else

```
pragma solidity ^0.5.0; contract ifelsedemo
{
uint i=10;
function decision_making() public view returns(string memory)
{
if(i%2==0)
{
return "even";
}
else
{
return "Odd";
}}}
```

For loop :

```
pragma solidity ^0.5.0;
contract loopDemo
{

uint [] data;


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

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

return data;

}

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

{
return data;
}
}
```

Do-while loop :

```
pragma solidity ^0.5.0;
// Creating a contract contract DoWhile {
// Declaring a dynamic array uint256[] data;

// Declaring state variable uint8 j = 0;

// Defining function to demonstrate
// 'Do-While loop'

contract DoWhile {
uint256[] data;

uint8 j = 0;

function loop() public returns (uint256[] memory) { do {
j++;
data.push(j);
} while (j < 5); return data;
}
function display() public view returns(uint256[] memory){ return data;}}
```

While loop :

```
pragma solidity ^0.5.0;
contract whiledemo
{
uint [] data; uint x=0;
function whileLoopDemo() public
{
while(x<5)
{
data.push(x); x=x+1;
}
}
function dispwhileloop() public view returns(uint[] memory)
{
return data;
}
}
```

Enums

```
pragma solidity ^0.5.0;

contract enumdemo { enum week_days {
Monday, Tuesday,
Wednesday, Thursday, Friday,
Saturday, Sunday
}

week_days week;
week_days choice;
week_days constant default_value = week_days.Sunday;

function set_value() public {
choice = week_days.Tuesday;
}

function get_choice() public view returns (week_days) { return choice;
}

function get_defaultvalue() public pure returns (week_days) { return default_value;
}
}
```

Structs :

```
pragma solidity ^0.5.0;

contract structdemo { struct Book {
string name;
string author; uint256 id;
bool availability;
}
Book book2;
Book book1 = Book("A Little Life", "Hanya Yanagihara", 2, false);
function set_details() public {
book2 = Book("Almond", "Sohn won-pyung", 1, true);
}

function book_info() public
view
returns (
string memory, string memory, uint256,
bool
)
{
return (book1.name, book1.author, book1.id, book1.availability);
}

function get_details() public
view
returns (
string memory, string memory, uint256, bool
)
{
return (book2.name, book2.author, book2.id, book2.availability);
}
}
```

9. Mappings

```
pragma solidity ^0.5.0;

contract LedgerBalance {
mapping(address => uint256) public balances;

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

contract Updater {
function updateBalance() public returns (uint256) {
LedgerBalance ledgerBalance = new LedgerBalance(); return
ledgerBalance.balances(address(this));
}}
```

10. Conversions

```
// SPDX-License-Identifier: MIT
pragma solidity  ^0.8.0;
contract ImplicitConversion {
function add() public pure returns (uint256) {
```

```
uint256 a = 10;
uint256 b = 20;
return a + b;
}
}
contract ExplicitConversion {
function convert() public pure returns (bytes memory) {
string memory str = "Hello World";
bytes memory b = bytes(str);
return b;
}
}
```

11.Ether Units

```
// SPDX-License-Identifier: MIT
pragma solidity  ^0.8.0;
contract SolidityTest {
function convert_Amount_to_Wei(uint256 Amount)
public
pure
returns (uint256)
{
return Amount * 1 wei;
}
function convert_Amount_To_Ether(uint256 Amount)
public
pure
returns (uint256)
{
return Amount * 1 ether;
}
function convert_Amount_To_Gwei(uint256 Amount)
public
pure
returns (uint256)
{
return Amount * 1 gwei;
}
function convert_seconds_To_mins(uint256 _seconds)
public
pure
returns (uint256)
{
return _seconds / 60;

}
function convert_seconds_To_Hours(uint256 _seconds)
public
pure
returns (uint256)
{
return _seconds / 3600;
}
function convert_Mins_To_Seconds(uint256 _mins)
public
pure
returns (uint256)
{
return _mins * 60;
}
}
```

Special Variables :

```
// SPDX-License-Identifier: MIT
pragma solidity  ^0.8.0;
contract Special_Variables {
mapping(address => uint256) rollNo;
function setRollNO(uint256 _myNumber) public {
rollNo[msg.sender] = _myNumber;
}
function whatIsMyRollNumber() public view returns (uint256) {
return rollNo[msg.sender];
}
}
```

B)Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions

Functions :

```
// SPDX-License-Identifier: MIT
pragma solidity  ^0.5.0;
contract view_demo {
uint256 num1 = 2;
uint256 num2 = 4;
function getResult() public view returns (uint256 product, uint256 sum) {
product = num1 * num2;
sum = num1 + num2;
}
}
```

2. Pure Functions

```
// SPDX-License-Identifier: MIT
pragma solidity  ^0.5.0;
contract pure_demo {
function getResult() public pure returns (uint256 product, uint256 sum) {
uint256 num1 = 2;
uint256 num2 = 4;
product = num1 * num2;
sum = num1 + num2;
}
}
```

3. Mathematical Functions

```
pragma solidity ^0.5.0;
contract Test{
function CallAddMod() public pure returns(uint){
return addmod(7,3,3);
}
function CallMulMod() public pure returns(uint){
return mulmod(7,3,3);
}
}
```

4. Cryptographic Functions :

```
pragma solidity ^0.5.0;
contract Test{
function callKeccak256() public pure returns(bytes32 result){
return keccak256("BLOCKCHAIN");
}
function callsha256() public pure returns(bytes32 result){
return sha256("BLOCKCHAIN");
}
function callripemd() public pure returns (bytes20 result){
return ripemd160("BLOCKCHAIN");
}
}
```

5. Functions :

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;


contract Test {
function return_example()
public
pure
returns (
uint256,
uint256,
uint256,
string memory
)
{
uint256 num1 = 10;
uint256 num2 = 16;
uint256 sum = num1 + num2;
uint256 prod = num1 * num2;
uint256 diff = num2 - num1;
string memory message = "Multiple return values";
return (sum, prod, diff, message);
}
}
```

6. Fallback function

```
// SPDX-License-Identifier: MIT

pragma solidity  ^0.5.0;

contract A {
uint256 n;
function set(uint256 value) external {
n = value;
}
function() external payable {
n = 0;
}
}
contract example {
function callA(A a) public returns (bool) {
(bool success, ) = address(a).call(abi.encodeWithSignature("setter()"));
require(success);
address payable payableA = address(uint160(address(a)));
return (payableA.send(2 ether));
}
}
```

7. Function Overloading

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.5.0;
```

```
contract OverloadingExample {
function add(uint256 a, uint256 b) public pure returns (uint256) {
return a + b;
}
function add(string memory a, string memory b)
public
pure
returns (string memory)
{
return string(abi.encodePacked(a, b));
}
}
```

8. Function modifiers

```
// SPDX-License-Identifier: MIT
pragma solidity  ^0.5.0;
contract ExampleContract {
address public owner = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
uint256 public counter;
modifier onlyowner() {
require(msg.sender == owner, "Only the contract owner can call");
_;
}
function incrementcounter() public onlyowner {
counter++;
}
}
```

PRACTICAL-4 IMPLEMENT AND DEMONSTRATE THE USE OF THE

FOLLOWING IN SOLIDITY

A) Withdrawal Pattern, Restricted Access

1) Withdrawal Pattern

```
// SPDX-License-Identifier: MIT
pragma solidity  0.8.18;
contract WithdrawalPattern {
address public owner;
uint256 public lockedbalance;
uint256 public withdrawablebalance;
constructor() {
owner = msg.sender;
}
modifier onlyowner() {
require(msg.sender == owner, "Only the owner can call this function");
_;
}
function deposit(uint256 amount) public payable {
require(amount > 0, "Amount must be greater than zero");
lockedbalance += amount;
}
function withdraw(uint256 amount) public payable onlyowner {
require(
amount <= withdrawablebalance,
"Insufficient withdrawable balance"
);
withdrawablebalance -= amount;
payable(msg.sender).transfer(amount);
}
function unlock(uint256 amount) public onlyowner {
require(amount <= lockedbalance, "Insufficient locked balance");
lockedbalance -= amount;
withdrawablebalance += amount;
}
}
```

2) Restricted Access

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract RestrictedAccess {
address public owner = msg.sender;
uint256 public creationTime = block.timestamp;
modifier onlyBy(address _account) {
require(msg.sender == _account, "Sender not authorized!");
_;
}
modifier onlyAfter(uint256 _time) {
require(block.timestamp >= _time, "Function was called too early!");
_;
}
modifier costs(uint256 _amount) {
require(msg.value >= _amount, "Not enough Ether provided!");
_;
}
function forceOwnerChange(address _newOwner)
public
payable
costs(200 ether)
{
owner = _newOwner;
}
function changeOwner(address _owner) public onlyBy(owner) {
owner = _owner;
}
function disown() public onlyBy(owner) onlyAfter(creationTime + 3 weeks) {
delete owner;
}
}
```

B) Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces

1) Contracts

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract Contract_demo {
string message = "Hello";
function dispMsg() public view returns (string memory) {
return message;
}
}
```

2) Inheritance

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract Parent {
uint256 internal sum;
function setValue() external {
uint256 a = 10;
uint256 b = 20;
sum = a + b;
}
}
contract child is Parent {
function getValue() external view returns (uint256) {
return sum;
}
}
contract caller {
child cc = new child();
function testInheritance() public returns (uint256) {
cc.setValue();
return cc.getValue();
}
function show_value() public view returns (uint256) {
return cc.getValue();
}
}
```

3) Abstract Contracts

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
abstract contract Calculator {
function getResult() external pure virtual  returns (uint256)  ;
}
contract Test is Calculator {
constructor()  {}
function getResult() external pure override  returns (uint256)            {
uint256 a = 1;
uint256 b = 2;
uint256 result = a + b;
return result;
}
}
```

4) Constructors

```
pragma solidity ^0.8.18;
contract constructorExample {
string str;
constructor() public {
str = "GeeksForGeeks";
}
function getValue() public view returns (string memory) {
return str;
}
}
```

5) Interfaces

```
pragma solidity ^0.8.18;
interface Calculator {
function getResult() external pure returns(uint);
}
contract Test is Calculator {
constructor()  {}
function getResult() external pure returns(uint){
uint a = 1;
uint b = 2;
uint result = a + b;
return result;
}
}
```

C) Libraries, Assembly, Events, Error handling.

1) Libraries

```
library myMathLib {
function sum(uint256 a, uint256 b) public pure returns (uint256) {
return a + b;
}
function exponent(uint256 a, uint256 b) public pure returns (uint256) {
return a**b;
}
}
```

using_library.sol Code

```
//SPDX-License-Identifier: MIT
pragma solidity  ^0.8.18;
import "contracts/variable.sol";
contract UseLib {
function getsum(uint256 x, uint256 y) public pure returns (uint256) {
return myMathLib.sum(x, y);
}
function getexponent(uint256 x, uint256 y) public pure returns (uint256) {
return myMathLib.exponent(x, y);
}
}
```

2) Assembly

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract InlineAssembly {
// Defining function
function add(uint256 a) public view returns (uint256 b) {
assembly {
let c := add(a, 16)
mstore(0x80, c)
{
let d := add(sload(c), 12)
b := d
}
b := add(b, c)
}
}
}
```

3) Events

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract eventExample {
// Declaring state variables
uint256 public value = 0;
// Declaring an event
event Increment(address owner);
// Defining a function for logging event
function getValue(uint256 _a, uint256 _b) public {
emit Increment(msg.sender);
value = _a + _b;
}
}
```

4) Error handling

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract ErrorDemo {
function getSum(uint256 a, uint256 b) public pure returns (uint256) {
uint256 sum = a + b;
// require(sum < 255, "Invalid");
assert(sum<255);
return sum;
}
}
```

PRACTICAL-5 WRITE A PROGRAM TO DEMONSTRATE MINING OF ETHER

```
const {Web3} = require('web3');

const web3 = new Web3(new Web3.providers.HttpProvider('http://127.0.0.1:7545'));

async function mine() {
const accounts = await web3.eth.getAccounts(); const coinbaseacc1 = accounts[0];
const coinbaseacc2 = accounts[1];
console.log(`Mining ether on Ganache with coinbase address:
${coinbaseacc1}`);

while (true) { try {
await web3.eth.sendTransaction({ from: coinbaseacc1,
to: coinbaseacc2, value: 50,
});
console.log(`Mined a new block!`);
} catch (err) { console.error(err);
}
}
}

mine();
```

Command to run : node file_name

PRACTICAL-6 DEMONSTRATE THE RUNNING OF THE BLOCKCHAIN

NODE

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

```
{
"config": {
"chainId": 3792,
"homesteadBlock": 0,
"eip150Block": 0,
"eip155Block": 0,
"eip158Block": 0
},
"difficulty": "2000",
"gasLimit": "2100000",
"alloc": {
"0x0b6C4c81f58B8d692A7B46AD1e16a1147c25299F": {
"balance": "9000000000000000000"
}
}
}
```

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

Step 3-> Run command geth account new --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine
Step 4-> Run command geth --identity "localB" --http --http.port "8280" --http.corsdomain "*" --http.api "db,eth,net,web3" --datadir "C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine" --port "30303" --nodiscover --networkid 5777 console. This command will enable geth console.

Step 5-> Run the command miner.setEtherbase('0xC050FE4d9bAc591d29538e2FD9cCA848B29489D0’) in the geth console
Step 6-> Run the command miner.start() to start mining

Step 7-> Below screenshots are the mining processes running on your local Machine.

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

PRACTICAL-7 CREATE YOUR OWN BLOCKCHAIN AND DEMONSTRATE

ITS USE

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

```
const SHA256 = require("crypto-js/sha256"); class Block {
        constructor(index, timestamp, data, previousHash = "") { this.index = index;
        this.timestamp = timestamp; this.data = data; this.previousHash = previousHash;
this.hash = this.calculateHash();
        }

        calculateHash() { return SHA256(
        this.index + this.previousHash + this.timestamp + JSON.stringify(this.data)
        ).toString();
        }
        }

        class Blockchain { constructor() {
        this.chain = [this.createGenesisBlock()];
        }

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

        getLatestBlock() {
        return this.chain[this.chain.length - 1];
        }

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

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

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

```
            if (currentBlock.hash != currentBlock.calculateHash()) {
            return false;
}

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

return true;
}
}


let myCoin = new Blockchain();
myCoin.addBlock(new Block(1, "22/04/2023", { amount: 4 })); myCoin.addBlock(new
Block(2, "22/04/2023", { amount: 8 }));
 console.log('Is blockchain valid? ' + myCoin.isChainValid());
console.log(JSON.stringify(myCoin, null, 4));
```

Flow of execution
Step 1-> Make sure you have installed nodejs in your system

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

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

Aim : Install hyperledger fabric and composer. Deploy and execute the application.

1. Download VMware Player, you can use virtualbox as an alternate.
2. Download Ubuntu ISO
3. Install vmware player
4. Create VM of Ubuntu using vmware player

Prepare the VM

```
$ sudo dpkg-reconfigure locales // choose en_US.UTF -8 if in doubt

$ sudo apt-get update

$ sudo apt-get upgrade
```

Install pre-requists

```
$ sudo apt-get install curl git docker.io docker-compose golang nodejs npmInstall Docker

$ sudo usermod -a -G docker $USER
$ sudo systemctl start docker

$ sudo systemctl enable docker
$ sudo chmod 666 /var/run/docker.sock
```

Install Hyperledger Fabric

1. Check the latest version of fabric repository, at the time of writting this post, it is 1.4 2. Install Fabric
$ curl -sSL http://bit.ly/2ysbOFE | bash -s 1.4.03. Check if fabric is installed, you should see big "END" once done

```
$ cd fabric-samples/first-network

$ ./byfn.sh generate
$ ./byfn.sh up
```

3. Check if fabric is installed, you should see big "END" once done

```
$ cd fabric-samples/first-network

$ ./byfn.sh generate
$ ./byfn.sh up
```

Install Composer

1. Create new user, when asked about the full name, use something different than the full name used of
the main user, to avoid confusion next time you are logging on.
Note: if you need to keep things with the same user, jump to step number 4 followed by step number 7
directly

```
$ sudo adduser playground
```

2. Set permission for the new user

```
sudo usermod -aG sudo playground
```

3. Login as the new user

```
su - playground
```

4. Install the prerequisites by getting and running the script from github. It will ask for the password of
“playground” account to proceed.

```
$ curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh
$ chmod u+x prereqs-ubuntu.sh
$ ./prereqs-ubuntu.sh
```

5. Logout and login with the new user to get things activated properly

```
$ exit

$ su - playground
```

6. Install components needed for running Hyperledger Fabric

```
$ curl -sSL http://bit.ly/2ysbOFE | bash -s 1.4.0
```

7. Install components needed for running Hyperledger Composer

```
$ npm install -g composer-cli composer-rest-server generator-hyperledger-composer yo
composerplayground
```

8. Start Composer

```
$ composer-playground
```