

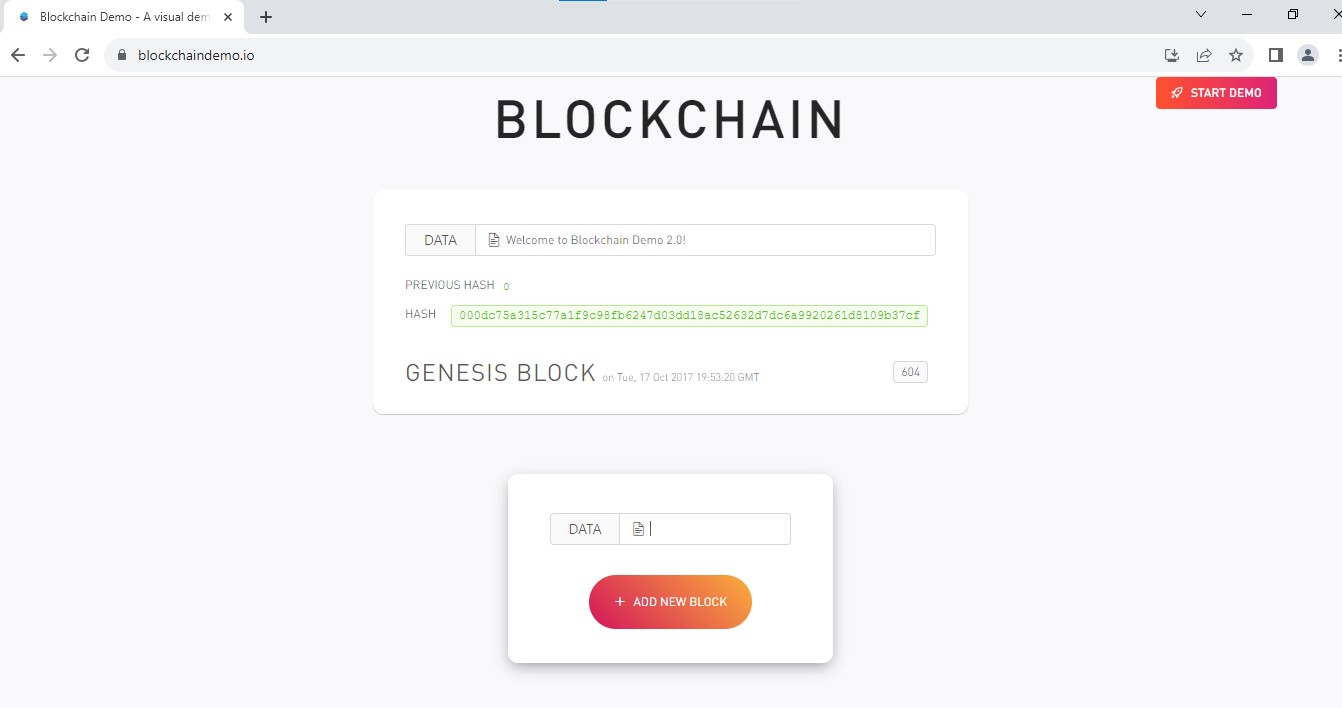


# PRACTICAL NO. 01-A

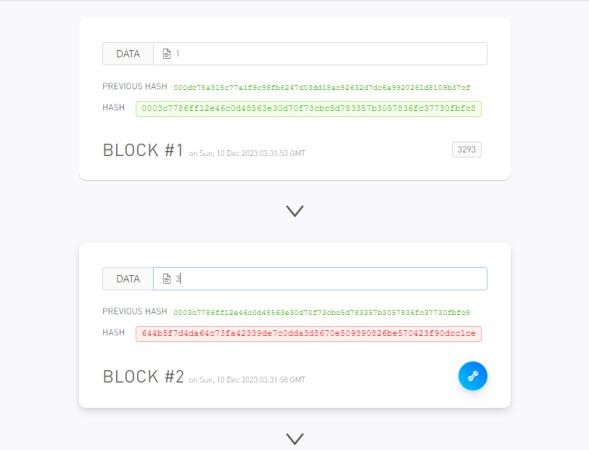
**Aim:** Perform Visual demonstration of Blockchain.

## Steps: A. blockchaindemo.io

1. Go to <https://blockchaindemo.io/>



1. Add data and - Add new blocks - Edit some middle block data to see the changes in the Blockchain

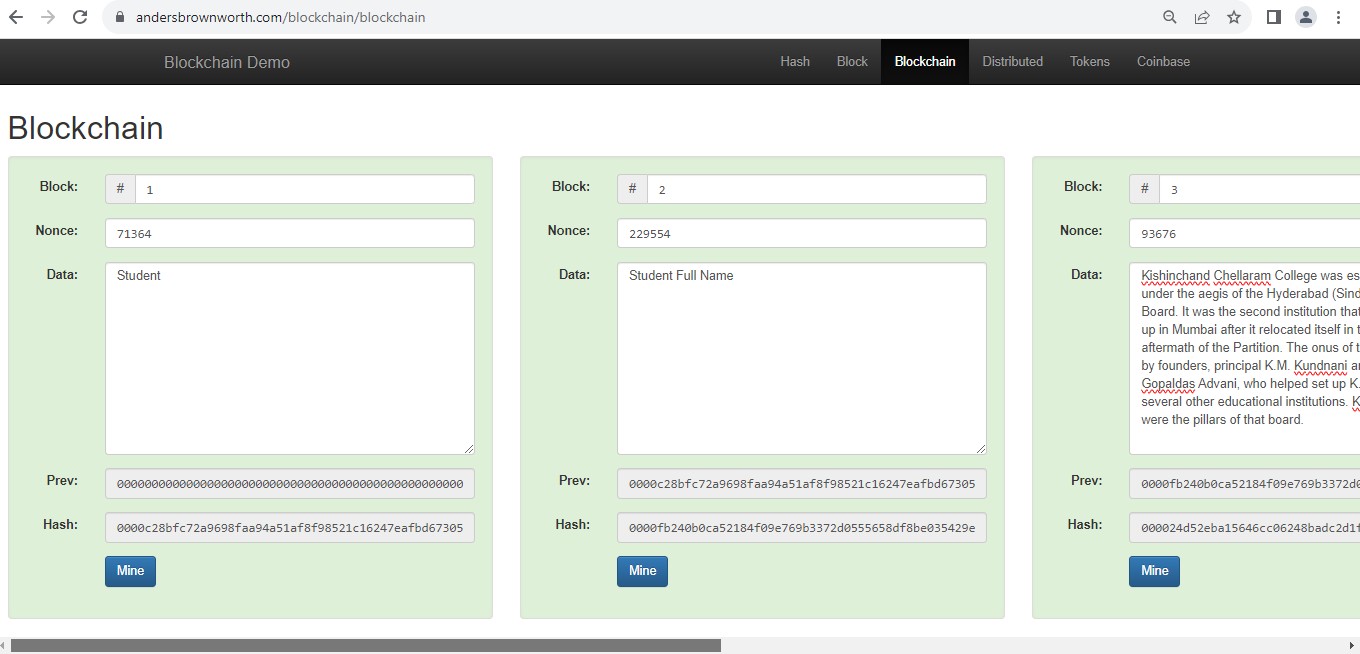




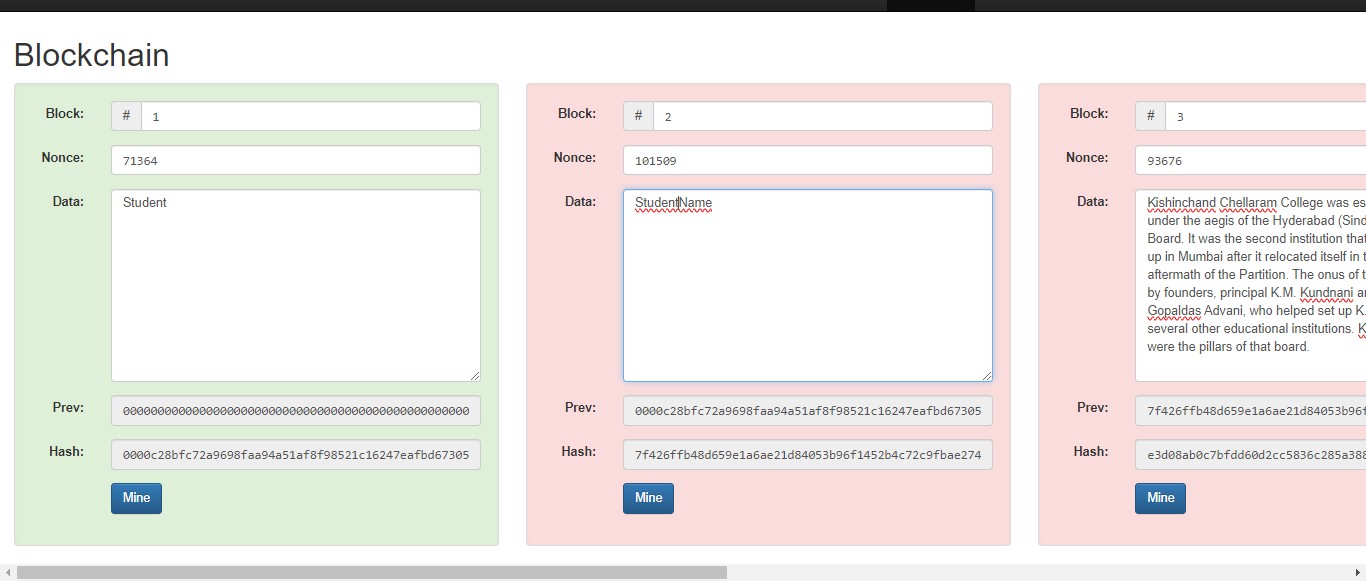


## B. andersbrownworth.com

1. Go to <https://andersbrownworth.com/blockchain/blockchain>



1. Add data and - Add new blocks - Edit some middle block data to see the changes in the Blockchain







# PRACTICAL NO. 01-B

**Aim:** Case Study on any application of Blockchain.

# UNRAVELING THE GENESIS OF BLOCKCHAIN

## A JOURNEY FROM CONCEPT TO REALITY

In the modern technological landscape, where innovation reigns supreme, few developments have captured the imagination and sparked as much curiosity as blockchain technology. To understand its existence and evolution, we must embark on a journey that traverses the realms of cryptography, decentralized networks, and digital currencies.

The story of blockchain begins with a fundamental problem: how to establish trust and facilitate secure transactions in a digital environment without the need for intermediaries. The concept of a distributed ledger, where information is stored across multiple nodes in a network, emerged as a potential solution to this challenge.

In the early 1990s, researchers and cryptographers toyed with the idea of creating a system that would enable secure timestamping of digital documents to prevent tampering and fraud. This led to the development of cryptographic techniques such as hash functions and Merkle trees, which laid the groundwork for the later evolution of blockchain.

However, it wasn't until the emergence of Bitcoin in 2009 that the concept of blockchain truly came into its own. Satoshi Nakamoto, the pseudonymous creator of Bitcoin, introduced blockchain as the underlying technology powering the world's first decentralized digital currency.

At its core, blockchain is a distributed ledger that records transactions in a transparent, immutable, and secure manner. Each block in the chain contains a cryptographic hash of the previous block, creating a chronological and tamper-proof record of all transactions on the network.

The key innovation of blockchain lies in its decentralized nature. Unlike traditional centralized systems where a single authority controls the flow of information, blockchain operates on a peer- to-peer network where every participant (or node) has a copy of the entire ledger. This ensures that no single entity has the power to manipulate or censor transactions, thereby fostering trust and transparency in digital interactions.

The advent of Bitcoin ignited a wave of enthusiasm and experimentation with blockchain technology. Developers and entrepreneurs recognized its potential to revolutionize not only the financial industry but also various other sectors such as supply chain management, healthcare, and voting systems.





As interest in blockchain grew, alternative cryptocurrencies and blockchain platforms began to emerge, each offering unique features and capabilities. Ethereum, launched in 2015 by Vitalik Buterin, introduced smart contracts, programmable agreements that automatically execute predefined actions when certain conditions are met. This innovation expanded the scope of blockchain beyond simple value transfer to include complex decentralized applications (dApps).

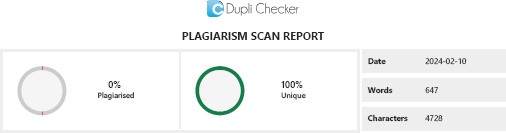
The proliferation of blockchain projects gave rise to a vibrant ecosystem of developers, investors, and enthusiasts, all seeking to harness the transformative power of this technology. Despite the hype and speculation surrounding blockchain, skeptics raised concerns about scalability, energy consumption, and regulatory challenges.

Nevertheless, the momentum behind blockchain continued to build, fueled by high-profile partnerships, corporate endorsements, and government initiatives. Companies across industries began exploring blockchain use cases for streamlining operations, reducing costs, and enhancing security.

In recent years, advancements in blockchain technology have paved the way for exciting developments such as non-fungible tokens (NFTs), decentralized finance (DeFi), and interoperable blockchain networks. These innovations demonstrate the adaptability and resilience of blockchain, as it continues to evolve and shape the future of digital interactions.

Looking ahead, the future of blockchain appears promising yet uncertain. While some envision a world where blockchain underpins the infrastructure of the internet, others caution against unrealistic expectations and hype. The journey of blockchain from concept to reality is far from over, and its ultimate impact on society remains to be seen.

In conclusion, the existence of blockchain technology is a testament to human ingenuity and the relentless pursuit of innovation. What began as an obscure concept in cryptography has evolved into a global phenomenon with far-reaching implications. As we stand on the cusp of a blockchain- powered future, one thing is clear: the story of blockchain is still being written, and its full potential has yet to be realized.







# PRACTICAL NO. 02-A

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

## Code:

from Crypto.PublicKey import RSA key=RSA.generate(1024) p\_key=key.public\_key().export\_key("PEM") priv\_key=key.export\_key("PEM") print("Prachi Gunjaria KSMSCIT009 \n") print(p\_key)

print(priv\_key)

**Output:**







# PRACTICAL NO. 02-B

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

## Code:

import binascii import datetime

from Crypto import Random

from Crypto.PublicKey import RSA from Crypto.Cipher import PKCS1\_v1\_5 from Crypto.Hash import SHA1

from Crypto.Signature import PKCS1\_v1\_5 import collections

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, 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 = SHA1.new(str(self.to\_dict()).encode('utf8')) return binascii.hexlify(signer.sign(h)).decode('ascii')

def display\_tran(transaction): dict = transaction.to\_dict()

print('\n Sender: Prachi --> \n' + dict['sender']) print('\n Receiver: Gunjaria --> \n' + dict['receiver']) print('\n Value --> \n' + str(dict['value']))

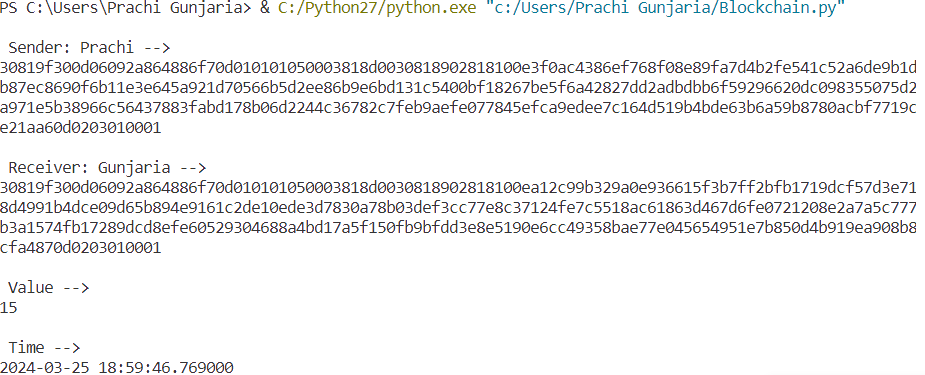
print('\n Time --> \n' + str(dict['time'])) transactions = []

Prachi=Client() Gunjaria=Client()

t1 = Transaction(Prachi, Gunjaria.identity, 15) t1.sign\_transaction()

display\_tran(t1)

**Output:**







# PRACTICAL NO. 03-A

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

## Code:

import binascii import datetime

from Crypto import Random

from Crypto.PublicKey import RSA from Crypto.Cipher import PKCS1\_v1\_5 from Crypto.Hash import SHA1

from Crypto.Signature import PKCS1\_v1\_5 import collections

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, 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 = SHA1.new(str(self.to\_dict()).encode('utf8')) return binascii.hexlify(signer.sign(h)).decode('ascii')

def display\_tran(transaction): dict = transaction.to\_dict()

print('Sender: --> ' + dict['sender']) print('Receiver: --> ' + dict['receiver']) print('Value: --> ' + str(dict['value']))

print('Time: --> ' + str(dict['time'])) transactions = []

Prachi=Client() Krishna=Client() Gunjaria=Client()

t1 = Transaction(Prachi, Gunjaria.identity, 15) t1.sign\_transaction()

transactions.append(t1)

t2 = Transaction(Prachi, Krishna.identity, 15) t2.sign\_transaction()

transactions.append(t2)

t3 = Transaction(Krishna, Gunjaria.identity, 15) t3.sign\_transaction()

transactions.append(t3)

tn = 1

for i in transactions: print("Transaction #", tn) display\_tran(i)

tn=tn+1

print(' \n')





**Output:**

|  |
| --- |
| Transaction # 1 Sender: -->  30819f300d06092a864886f70d010101050003818d0030818902818100d5ce4f844f1dd6d5ad1190a0  854cc78b0e98bb0052a60ee456e88441fbe4e39bbc318592517014e915096c663fca05541b9bc8f886 6823cb778c4b615f893f9f78439bf04ca12c3b8dd43dbe380cf65034bd0d230bd5aaf87c336a405c8b 39a48d2f7e0a2cdf17fa75e5484e7d5f7d364c3264c0af78c758e69abd44c3ba132b0203010001 Receiver: --> 30819f300d06092a864886f70d010101050003818d0030818902818100b76b254a59406c3036ddeda9  60392d1c66835d5a8a4df9d30b45731cbad9dcc34116c753dfafa6fdf6b04904346f84ee0fc9a838df f2bed9d02d402db3a45aadd7ed95d1299f0cf0b7b9eaa6b16b9e08913d0b521289663bc55b8091fd4a 737318f106a904c967bc53043b4b81899db91c419216d83ad91514b0943982d3bab10203010001  Value: --> 15  Time: --> 2024-01-14 09:01:48.673846 |
| Transaction # 2 Sender: -->  30819f300d06092a864886f70d010101050003818d0030818902818100d5ce4f844f1dd6d5ad1190a0  854cc78b0e98bb0052a60ee456e88441fbe4e39bbc318592517014e915096c663fca05541b9bc8f886 6823cb778c4b615f893f9f78439bf04ca12c3b8dd43dbe380cf65034bd0d230bd5aaf87c336a405c8b 39a48d2f7e0a2cdf17fa75e5484e7d5f7d364c3264c0af78c758e69abd44c3ba132b0203010001 Receiver: --> 30819f300d06092a864886f70d010101050003818d00308189028181009e84bd59d5fe7d0b6bb84a60  554dcc062d33ebf3d8a765b19d5b3584409ec40071ddf262abc8763b1216058c25969fb8749c133e2d 80e9f780175e6302fed990eda53e70f3615390962fc7161bd166b06d1b776d0070da457b39302ae52e e505e17728f87b88bb0f78cad78552d225a74677f7d5af650edd7a09bea1326025530203010001  Value: --> 15  Time: --> 2024-01-14 09:01:48.675845 |
| Transaction # 3 Sender: -->  30819f300d06092a864886f70d010101050003818d00308189028181009e84bd59d5fe7d0b6bb84a60  554dcc062d33ebf3d8a765b19d5b3584409ec40071ddf262abc8763b1216058c25969fb8749c133e2d 80e9f780175e6302fed990eda53e70f3615390962fc7161bd166b06d1b776d0070da457b39302ae52e e505e17728f87b88bb0f78cad78552d225a74677f7d5af650edd7a09bea1326025530203010001  Receiver: --> 30819f300d06092a864886f70d010101050003818d0030818902818100b76b254a59406c3036ddeda9  60392d1c66835d5a8a4df9d30b45731cbad9dcc34116c753dfafa6fdf6b04904346f84ee0fc9a838df f2bed9d02d402db3a45aadd7ed95d1299f0cf0b7b9eaa6b16b9e08913d0b521289663bc55b8091fd4a 737318f106a904c967bc53043b4b81899db91c419216d83ad91514b0943982d3bab10203010001  Value: --> 15  Time: --> 2024-01-14 09:01:48.676846 |





# PRACTICAL NO. 03-B

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

## Code:

import datetime import collections import binascii

from Crypto import Random

from Crypto.PublicKey import RSA from Crypto.Cipher import PKCS1\_v1\_5 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, receiver, amount): self.sender = sender

self.receiver = receiver self.amount = amount

self.timestamp = 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,





"amount": self.amount, "timestamp": self.timestamp

})

def sign\_tram(self):

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

h = SHA.new(str(self.to\_dict()).encode('utf-8')) return binascii.hexlify(signer.sign(h)).decode('ascii')

def display\_transaction(transaction): dict = transaction.to\_dict() print("Sender: ", dict["sender"])

print(" ")

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

print("Amount: ", dict["amount"])

print(" ")

print("Timestamp: ", dict["timestamp"]) 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)

class Block:

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





Prachi=Client()

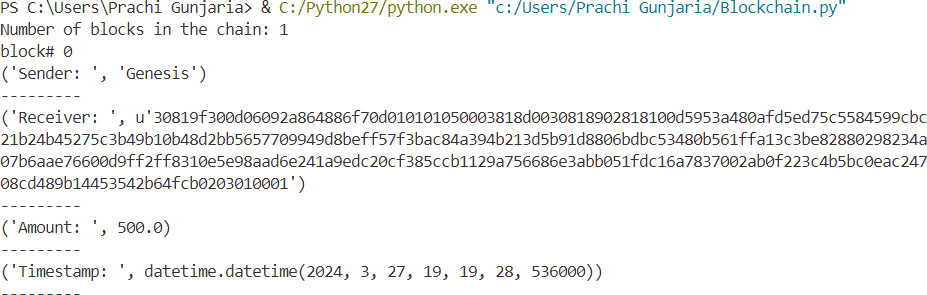
t0 = Transaction("Genesis", Prachi.identity,500.0)

block0=Block() block0.previous\_block\_hash=None Nounce=None

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

TPCoins=[] TPCoins.append(block0) dump\_blockchain(TPCoins)

**Output:**







# PRACTICAL NO. 04-A

**Aim:** 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 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 nounce: "+digest) return i

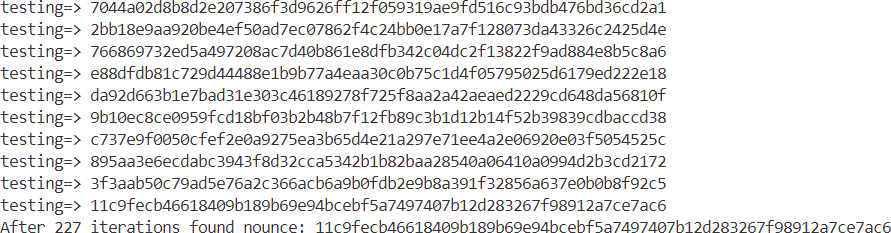
mine("Prachi",2)

## Output:



**…**

**…**







# PRACTICAL NO. 04-B

**Aim:** Add blocks to the miner and dump the blockchain.

## Code:

import hashlib import json

from time import time

class Blockchain: def init (self):

self.chain =[] self.pending\_transactions = []

### #Create a genesis block

self.add\_block(previou\_hash='1')

def add\_block(self,proof=None, previou\_hash=None): block = {

'index' : len(self.chain) + 1, 'timestamp' : time(),

'transactions' : self.pending\_transactions,

'proof' : proof or 0, ***#Proof of work***

'previous\_hash' : previous\_hash or self.hash(self.chain[-1]) if self.chain else None

}

***#Reset the list of pending transactions*** self.pending\_transactions = [] self.chain.append(block)

return block

def add\_transactions(self, sender, receipient, amount): transaction = {

'sender’ : sender, 'receipient’ : receipient, 'amount' : amount,

}

self.pending\_transactions.append(transaction) return self.last\_block['index'] + 1





@staticmethod def hash(block):

### # Hashes a block

block\_string=json.dumps(block, sort\_keys=True).encode() return hashlib.sha256(block\_string).hexdigest()

@property

def last\_block(self): return self.chain[-1]

my\_blockchain = Blockchain()

### # Add transactions

my\_blockchain.add\_transactions('Prachi', 'Manisha', 10)

my\_blockchain.add\_transactions('Manisha','Vaibhavi', 5)

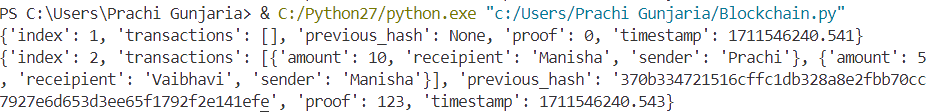
proof\_of\_work =123

previous\_hash = my\_blockchain.hash(my\_blockchain.last\_block) my\_blockchain.add\_block(proof\_of\_work, previous\_hash)

### #Dump the blockchain

for block in my\_blockchain.chain: print(block)

**Output:**







# PRACTICAL NO. 05

**Aim:** Implement and demonstrate the user of the following in Solidity:

## Variable

**Code:**

*// SPDX-License-Idewntifier: MIT*

pragma solidity ^0.8.0;

contract SolidityTest {

uint256 storedData; ***// State variable***

constructor() public { storedData = 10;

}

function getSum() public view returns (uint256) { uint256 a = 10; ***// local variable***

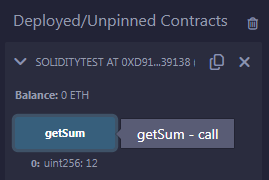
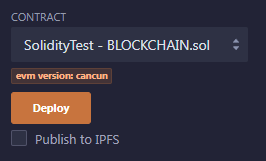
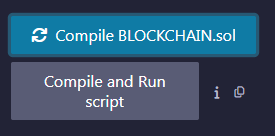
uint256 b = 2; uint256 result = a + b;

return result; ***// access the state variable***

}

}

## Output:







1. **Operations**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.0;

contract SolidityTest {

uint256 storedData; // State variable

constructor() public { storedData = 10;

}

function getDiv() public view returns (uint256) { uint256 a = 10; // local variable

uint256 b = 2; uint256 result = a / b;

return result; // accesss the state variable

}

function getMul() public view returns (uint256) { uint256 a = 10; // local variable

uint256 b = 2; uint256 result = a \* b;

return result; // accesss the state variable

}

function getSum() public view returns (uint256) { uint256 a = 10; // local variable

uint256 b = 2; uint256 result = a + b;

return result; // accesss the state variable

}





function getSub() public view returns (uint256) { uint256 a = 10; // local variable

uint256 b = 2; uint256 result = a - b;

return result; // accesss the state variable

}

}

## Output:





1. **Loops**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.0;

contract ForTest { uint256 result = 0;

function sum() public returns (uint256 data) { for (uint256 i = 11; i < 20; i++) {

result = result + i;

}

return result;

}

}

## Output:





1. **Decision Making**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract NumberChecker {

function checkEvenOdd(uint256 \_number) public view returns (string memory) { if (\_number % 2 == 0) {

return "Even";

} else {

return "Odd";

}

}

}

## Output:





1. **Strings**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.0;

contract SS {

string str1 = "M.SC.IT PART 2";

string str2 = "KC. COLLEGE COLABA"; string str3 = new string(20);

function getstr1() public returns (string memory) { return str1;

}

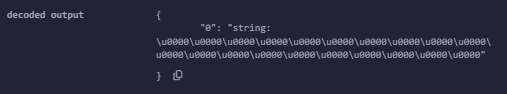
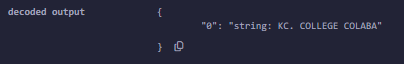
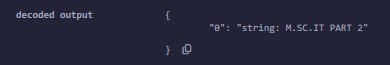
function getstr2() public returns (string memory) { return str2;

}

function getstr3() public returns (string memory) { return str3;

}

}

**Output:**





# PRACTICAL NO. 06

**Aim:** Implement and demonstrate the user of the following in Solidity:

## Arrays

**Code:**

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.8.0; contract MyArray {

uint256[] nums = [1, 2, 33, 21]; event PrintArray(uint256[] values);

function getLength() public view returns (uint256) { return nums.length;

}

function pop() public {

require(nums.length > 0, "Array is empty"); nums.pop();

}

function push(uint256 i) public { nums.push(i);

emit PrintArray(nums);

}

}

## Output:





1. **Enums**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract Types { enum WeekDays {

Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

}

WeekDays public week; WeekDays public choice;

WeekDays constant default\_value = WeekDays.Sunday; function set\_value() public {

choice = WeekDays.Thursday;

}

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

}

function get\_default\_value() public pure returns (WeekDays) { return default\_value;

}

}

## Output:





1. **Mappings**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract LedgerBalance { mapping(address => uint256) balance; mapping(address => string) name;

function updateBalance() public returns (uint256) { balance[msg.sender] = 20;

return balance[msg.sender];

}

function senderInfo() public returns (string memory) { name[msg.sender] = "Prachi";

return name[msg.sender];

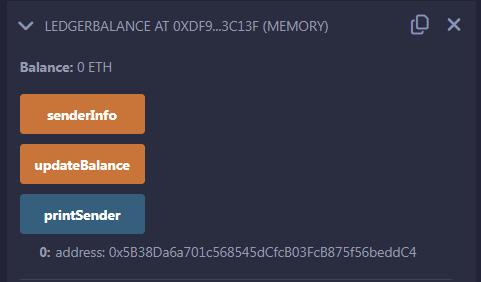
}

function printSender() public view returns (address) { return msg.sender;

}

}

## Output:







1. **Ether Units**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract EtherUnitsExample {

uint256 public valueInWei = 1 ether; // 1 ether in Wei uint256 public valueInFinney = 1 finney; // 1 finney in Wei uint256 public valueInSzabo = 1 szabo; // 1 szabo in Wei uint256 public valueInEther = 1 ether; // 1 ether in Wei

function convert(uint256 \_amount, string memory \_unit) public

pure

returns (uint256)

{

if (

keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("wei"))

) {

return \_amount;

} else if (

keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("finney"))

) {

return \_amount \* 1 finney;

}





else if (

keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("szabo"))

) {

return \_amount \* 1 szabo;

} else if (

keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("ether")) || keccak256(abi.encodePacked(\_unit)) == keccak256(abi.encodePacked("eth"))

) {

return \_amount \* 1 ether;

} else {

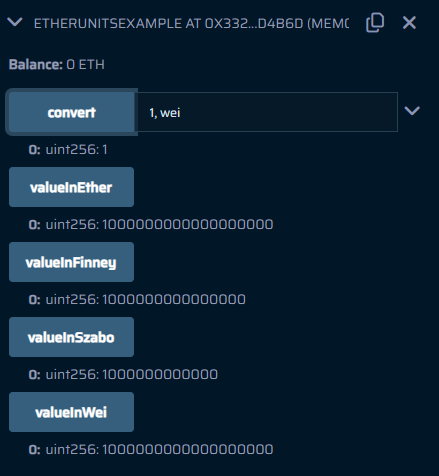
revert("Invalid unit");

}

}

}

## Output:







1. **Special Variables**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract LedgerBalancee { mapping(address => string) name;

function updateBalance() public returns (string memory) { name[msg.sender] = "Prachi";

return name[msg.sender];

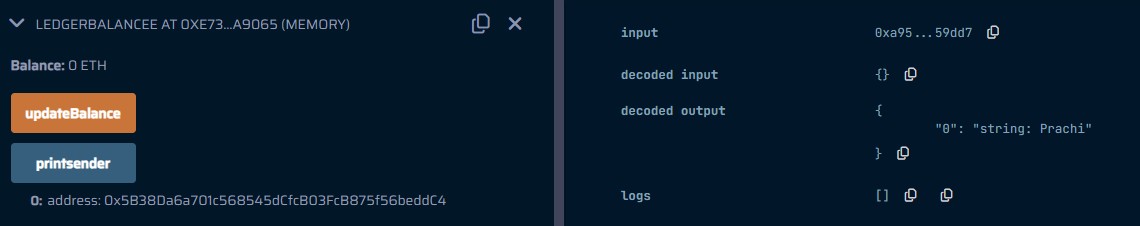
}

function printsender() public view returns (address) { return msg.sender;

}

}

**Output:**







# PRACTICAL NO. 07

**Aim:** Implement and demonstrate the user of the following in Solidity:

## View Functions

**Code:**

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract Test {

function getResult() public view returns (uint256 product, uint256 sum) { uint256 a = 1; // local variable

uint256 b = 2; product = a \* b; sum = a + b;

}

}

## Output:







1. **Pure Functions**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract Test {

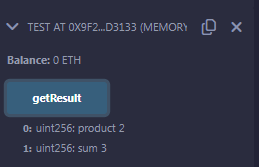
function getResult() public pure returns (uint256 product, uint256 sum) { uint256 a = 1; // local variable

uint256 b = 2; product = a \* b; sum = a + b;

}

}

## Output:







1. **Fallback Functions**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity 0.8.24;

contract Test {

string public calledFallbackfun;

fallback() external payable {

calledFallbackfun = "I am Fallback Function";

}

function getBalance() public view returns (uint256) { return address(this).balance;

}

}

contract Sender {

function transfer() public payable {

require(msg.value >= 2 ether, "Insufficient ether sent");

(bool sent, ) = payable(0x9d83e140330758a8fFD07F8Bd73e86ebcA8a5692)

.call{value: 5 ether}("Transaction Done"); require(sent, "transaction Failed");

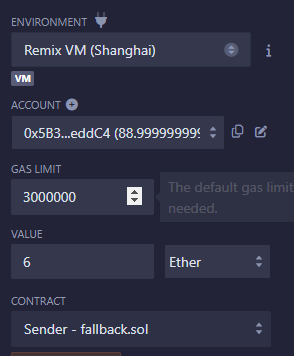
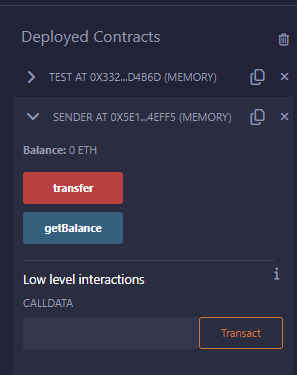
}

function getBalance() public view returns (uint256) { return address(this).balance;

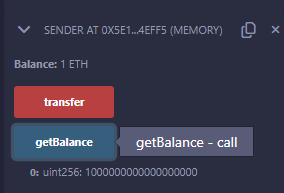
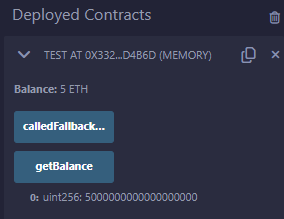
}

}

## Output:



Balance: 0 ETH Balance: 6 Ether







2. Test.getBalance ()

1. Sender.transfer()

3. Sender.getBalance ()





## Function Overloading

**Code:**

*// SPDX-License-Identifier: MIT* pragma solidity >=0.8.0 <0.9.0; contract Test {

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

}

function getsum( uint256 x, uint256 y, uint256 z

) public pure returns (uint256) { return x + y + z;

}

function callTwo() public pure returns (uint256) { return getsum(2, 4);

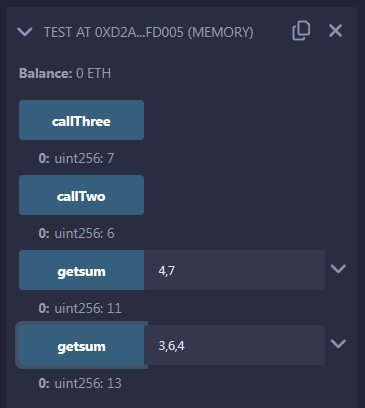
}

function callThree() public pure returns (uint256) { return getsum(2, 4, 1);

}

}

## Output:







1. **Mathematical Functions**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.12;

contract Test {

function callAddMod() public pure returns (uint256) { return addmod(4, 5, 3);

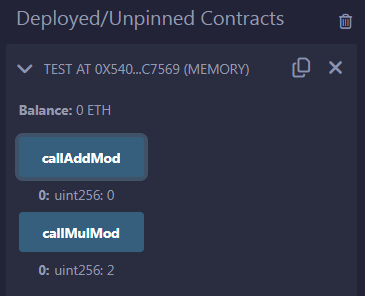
}

function callMulMod() public pure returns (uint256) { return mulmod(3, 4, 5);

}

}

## Output:







1. **Cryptographic Functions**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.12;

contract Test {

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

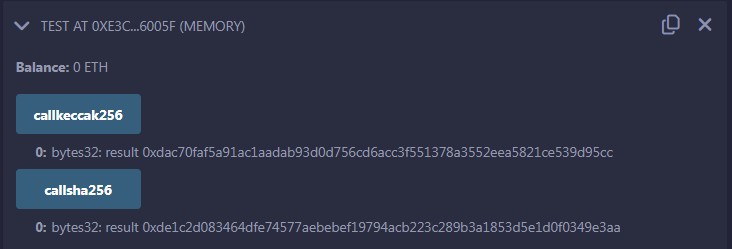
}

function callkeccak256() public pure returns (bytes32 result) { return keccak256("Prachi");

}

}

## Output:







1. **Generate Random Number**

## Code:

*// SPDX-License-Identifier: MIT* pragma solidity ^0.8.24; contract Randomnumber{

uint randomno = 0;

function setnumber() public {

randomno =uint (keccak256(abi.encodePacked(msg.sender,randomno)));

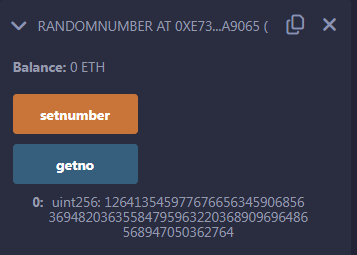
}

function getno() public view returns(uint){ return randomno;

}

}

**Output:**







# PRACTICAL NO. 08

**Aim:** Implement and demonstrate the user of the following in Solidity:

## Contracts

**Code:**

*// SPDX-License-Identifier: MIT*

pragma solidity 0.8.24; contract Test {

function TakeLoan(uint256) external payable { GiveLoan loan = new GiveLoan(350);

}

}

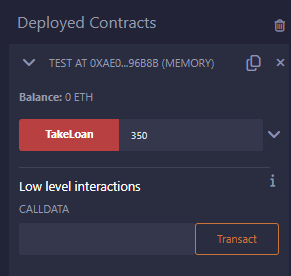
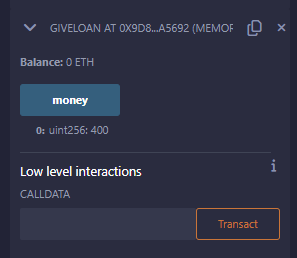
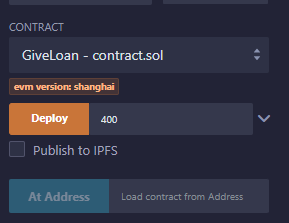
contract GiveLoan { uint256 public money;

constructor(uint256 \_amt) public { money = \_amt;

}

}

## Output:







1. **Inheritance**

## Single Inheritance

**Code:**

*//SPDX-License-Identifier:GPL-3.0*

pragma solidity 0.8.24;

contract Parent { uint256 internal sum;

function setVal() external { uint256 a = 50;

uint256 b = 20; uint256 c = 20; sum = a + b + c;

}

}

contract child is Parent {

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

}

}

contract caller {

child cc = new child();

function Inher() public returns (uint256) { cc.setVal();

return cc.getVal();

}

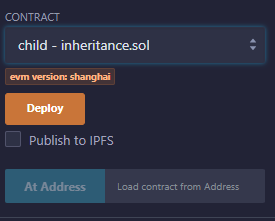
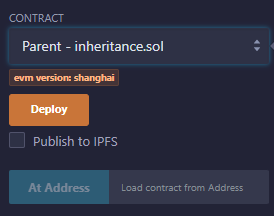
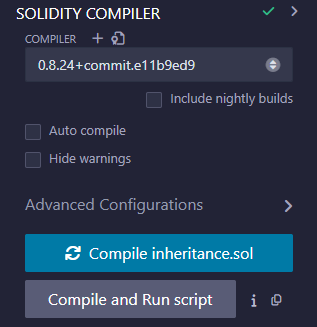
}

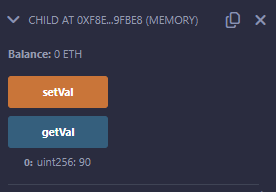
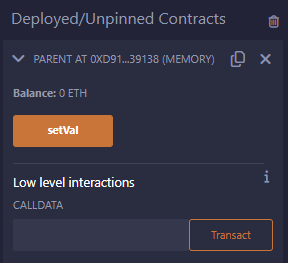


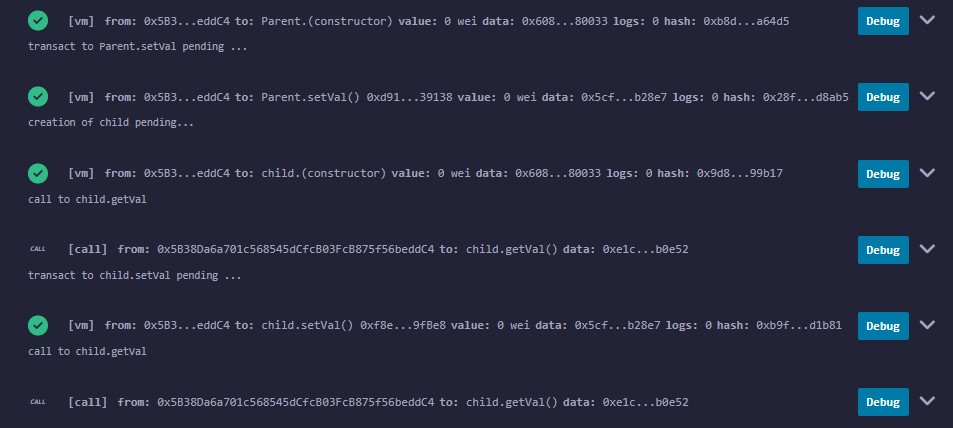


## Output:\

Compile and deploy:











## Multiple Inheritance

**Code:**

*//SPDX-License-Identifier:GPL-3.0*

pragma solidity >=0.8.2 <0.9.0;

contract A {

string internal x;

function setA() external {

x = "Multiple inheritance";

}

}

contract B {

uint256 internal pow; function setB() external {

uint256 a = 20; uint256 b = 2; pow = a\*\*b;

}

}

contract C is A, B {

function getStr() external returns (string memory) { return x;

}

function getPow() external returns (uint256) { return pow;

}

}

contract caller {

C contractC = new C();

function testInheritance() public returns (string memory, uint256) { contractC.setA();

contractC.setB();

return (contractC.getStr(), contractC.getPow());

}

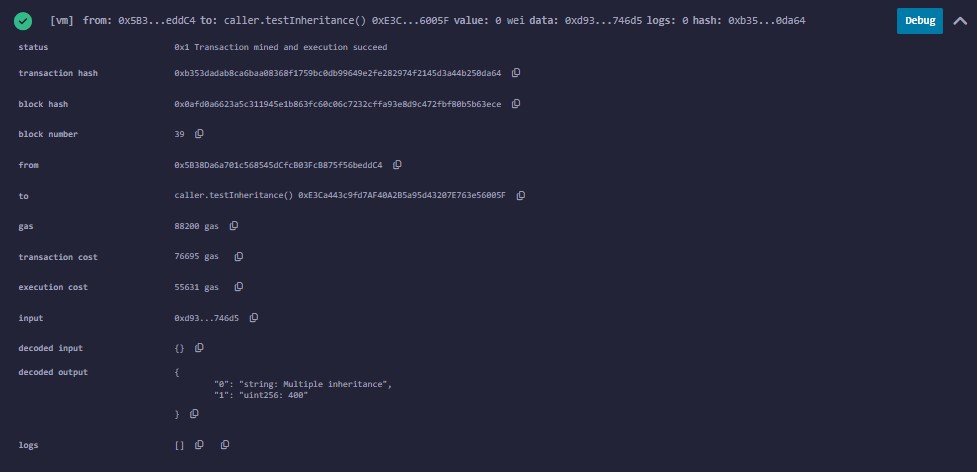
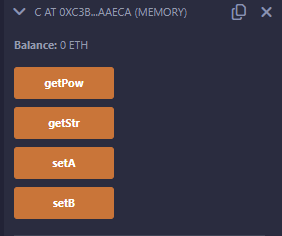
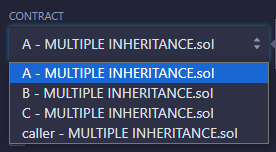
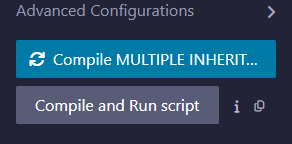
}

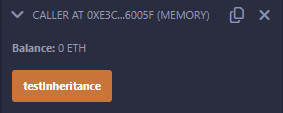
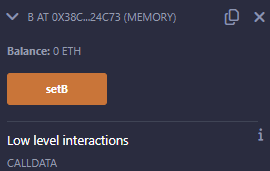
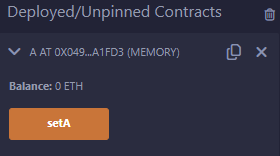




## Output:

Compile and deploy all contracts









## Constructors

**Code:**

*// SPDX-License-Identifier: MIT*

pragma solidity >=0.8.2 <0.9.0;

contract ConstructorExample { string str;

constructor() {

str = "K.C. College || Prachi Gunjaria || M.Sc. Part II - Roll No 09";

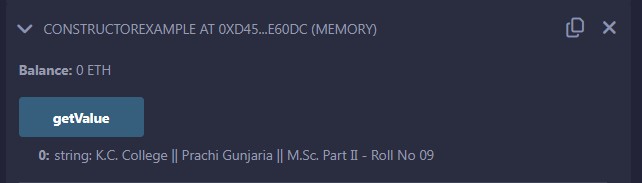
}

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

}

}

## Output:







1. **Abstract Class**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.17;

contract A {

function getValue() public view returns (uint256);

}

contract B is A {

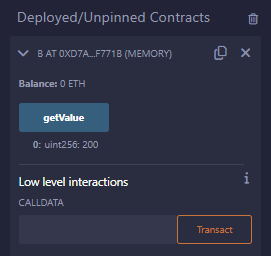
function getValue() public view returns (uint256) { uint256 x = 10;

uint256 y = 20; uint256 result = x \* y; return result;

}

}

## Output:







1. **Interface**

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.17;

interface A {

function getValue() external view returns (uint256);

}

contract B is A {

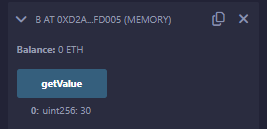
function getValue() external view returns (uint256) { uint256 x = 10;

uint256 y = 20; uint256 result = x \* y; return result;

}

}

**Output:**







# PRACTICAL NO. 09

**Aim:** Implement and demonstrate the user of the following in Solidity:

## Libraries

**Code:**

### File 01: MathUtils.sol

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0; library MathUtils {

function add(uint256 x, uint256 y) public pure returns (uint256) { return x + y;

}

}

### File 01: Library.sol

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0; import "./MathUtils.sol"; contract calculator {

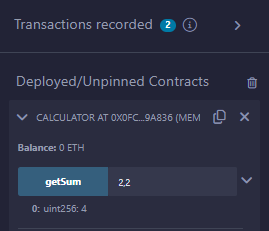
using MathUtils for uint256;

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

}

}

## Output:







1. **Error Handling**
2. Require

## Code:

*// SPDX-License-Identifier: MIT*

pragma solidity ^0.5.0;

contract requireStatement {

function checkInput(uint256 \_input) public view returns (string memory) { require(\_input >= 0, "invalid uint8");

require(\_input <= 255, "invalid uint8"); return "Input is Uint8";

}

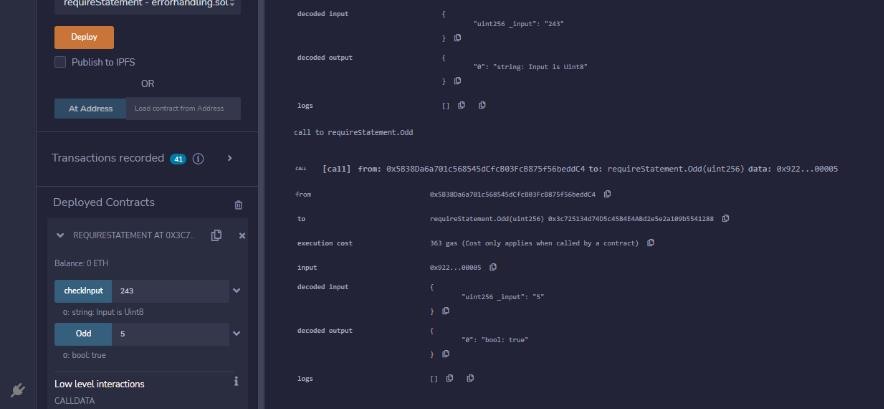
function Odd(uint256 \_input) public view returns (bool) { require(\_input % 2 != 0);

return true;

}

}

## Output:







1. Assert

## Code:

pragma solidity ^0.5.0; contract assertStatement {

bool result;

function checkOverflow(uint256 \_num1, uint256 \_num2) public { uint256 sum = \_num1 + \_num2;

assert(sum <= 255); result = true;

}

function getResult() public view returns (string memory) { if (result == true) {

return "No Overflow";

} else {

return "Overflow exist";

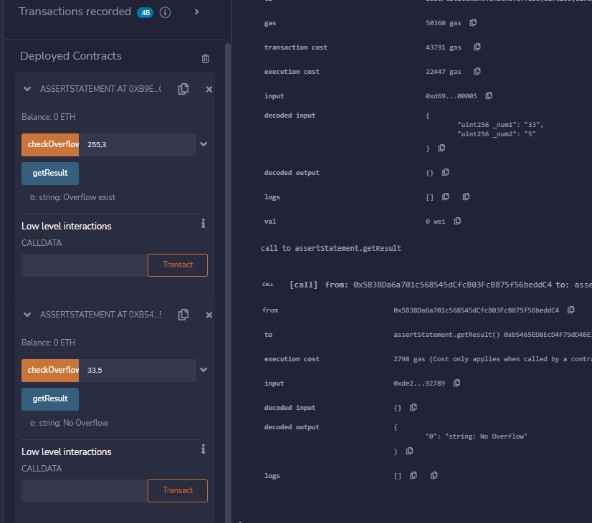
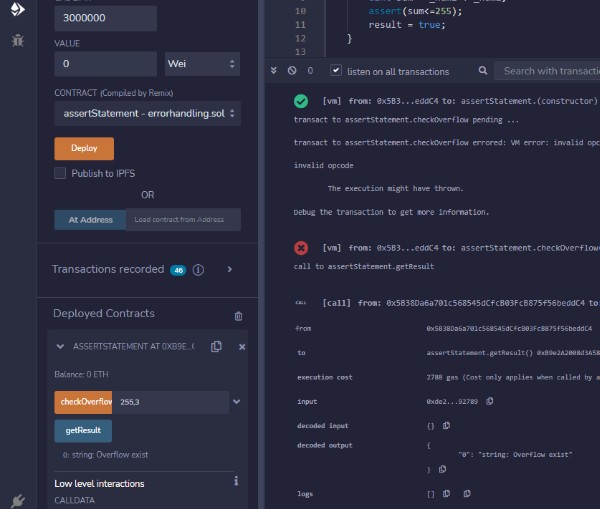
}

}

}

## Output:

### Overflow exist No Overflow







1. Revert

## Code:

pragma solidity ^0.5.0; contract revertStatement {

function checkOverflow(uint256 \_num1, uint256 \_num2)

public view

returns (string memory, uint256)

{

uint256 sum = \_num1 + \_num2; if (sum < 0 || sum > 255) {

revert(" Overflow Exist");

} else {

return ("No Overflow", sum);

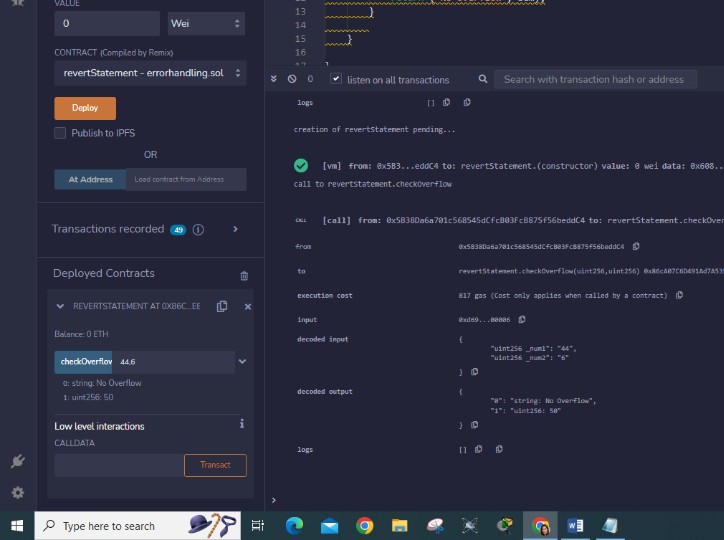
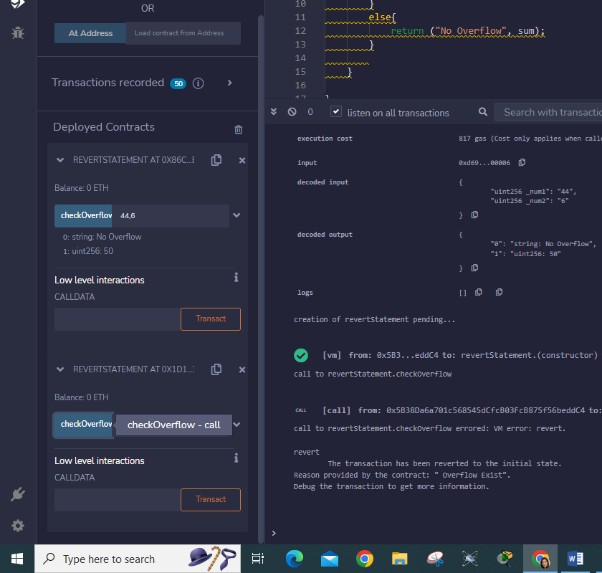
}

}

}

## Output:

***Overflow exist No Overflow***







# PRACTICAL NO. 10

**Aim:** Install Hyperledger Fabric and deploy and test it.

Steps to Install HyperLedger Fabric in Linux (Ubuntu)

1. Prerequisites
2. Linux Installations
3. Prerequisites
   * cURL - latest version
   * Docker - version 17.06.2-ce or greater
   * Docker Compose - version 1.14.0 or greater
   * Golang - version 1.11.x
   * Nodejs - version 8.x (other versions are not in support yet)
   * NPM - version 5.x
   * Python 2.7

*Note: These prerequisites’ versions are according to the fabric v1.4* documentation.

1. Linux(Ubuntu) installation steps
2. sudo apt-get install curl
3. sudo apt-get install golang-go
4. export GOPATH=$HOME/go
5. export PATH=$PATH:$GOPATH/bin
6. sudo apt-get install nodejs
7. sudo apt-get install npm
8. sudo apt-get install python
9. sudo apt-get install docker
10. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt key add -
11. sudo add-apt-repository “deb [arch=amd64] https://download.docker.com/linux/ubuntu

$(lsb\_release -cs) stable”

1. sudo apt-get update
2. apt-cache policy docker-ce
3. sudo apt-get install -y docker-ce
4. sudo apt-get install docker-compose
5. sudo apt-get upgrade

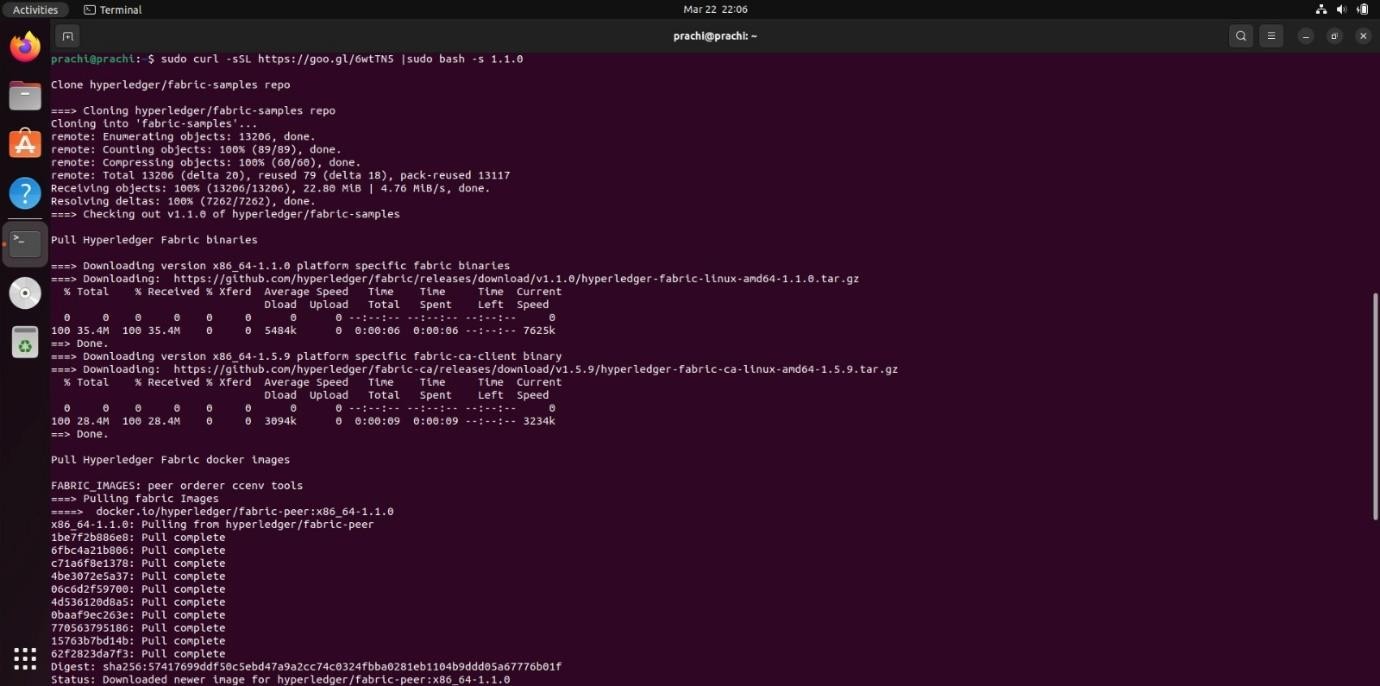
*With the help of the above 15 steps, our environment is set up.*

### Next, we’re going to download the samples of Fabric that have already been prepared to test it out. Enter the following two commands in your terminal.

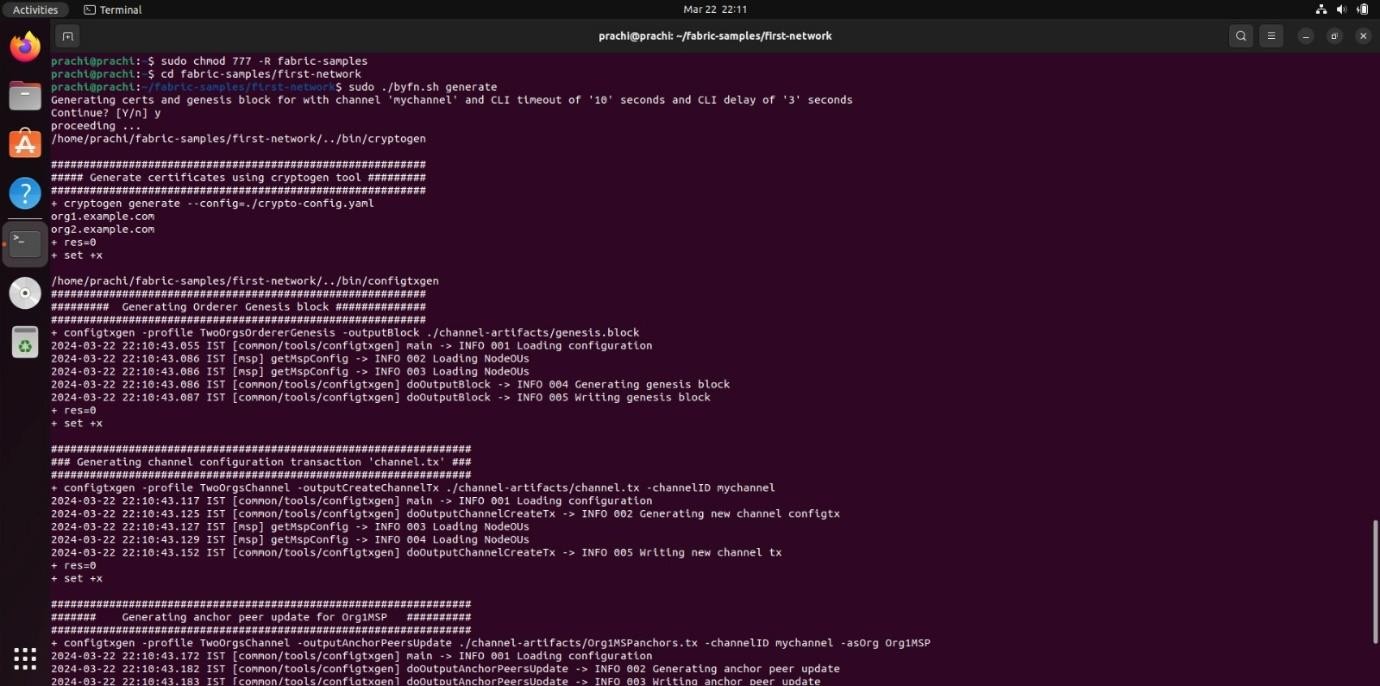




1. sudo curl -sSL https://goo.gl/6wtTN5 |sudo bash -s 1.1.0



1. sudo chmod 777 -R fabric-samples
2. cd fabric-samples/first-network
3. sudo ./byfn.sh generate

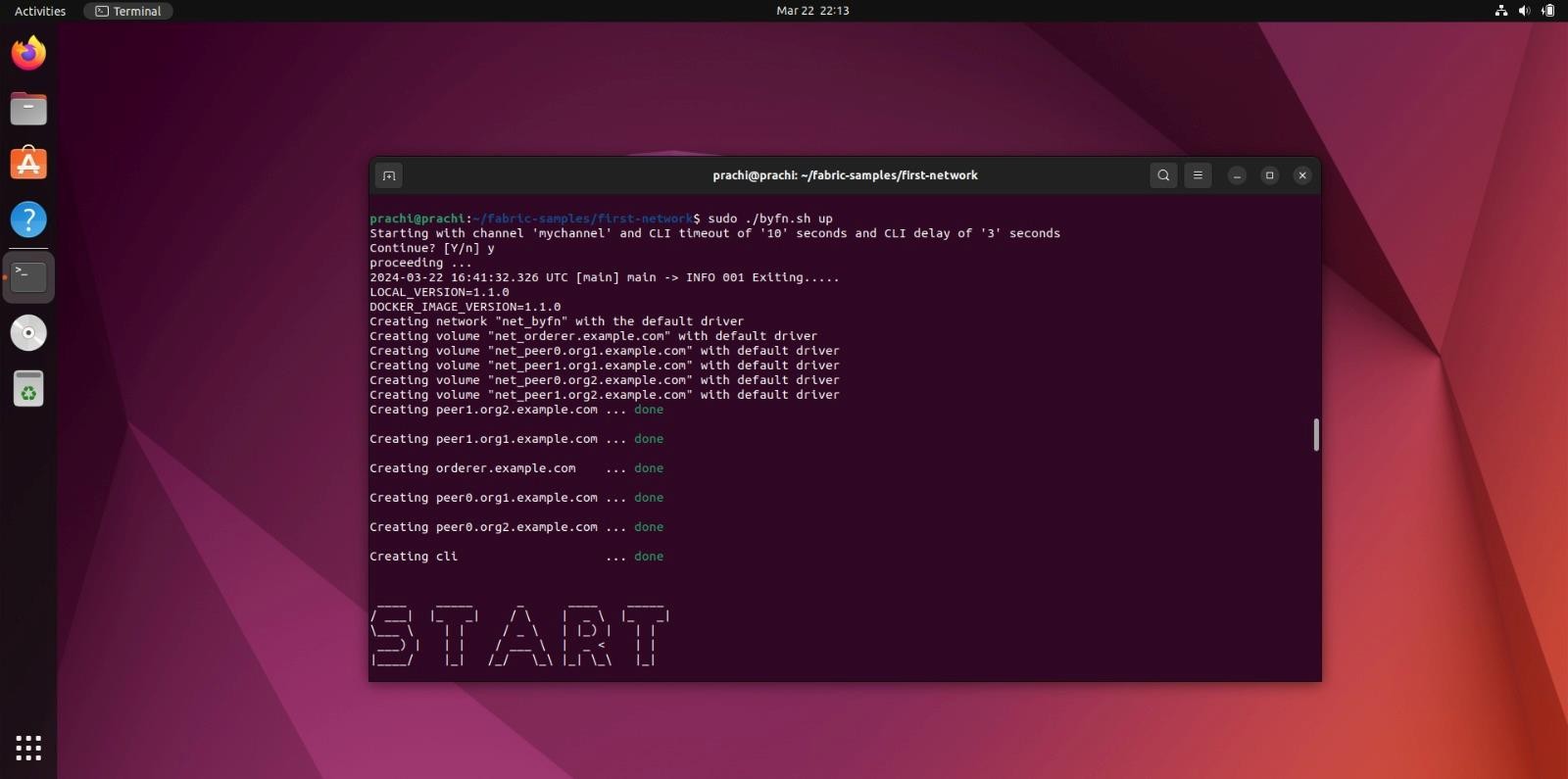






### Now bring the blockchain network up with the following command.

1. sudo ./byfn.sh up



If everything worked, then you successfully created your first Fabric network!

### Congratulations!

*For the time being you can bring the network down with the command:*

1. sudo ./byfn.sh down

Thus, we reached a state where our computer can successfully use and deploy Hyperledger Fabric blockchain networks.