Practical No. 1

A. Develop a secure messaging application where users can exchange messages securely using RSA encryption. Implement a mechanism for generating RSA key pairs and encrypting/decrypting messages.

Code:

RSA key generation. Or The one given below:

```
#pip install cryptography
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
def generate_rsa_key_pair():
"""Generates a new RSA public and private key pair."""
private_key = rsa.generate_private_key(
public_exponent=65537,
key_size=2048,
backend=default_backend()
)
public_key = private_key.public_key()
return private_key, public_key
def encrypt_message(public_key, message):
"""Encrypts a message using the recipient's public key."""
ciphertext = public_key.encrypt(
message.encode('utf-8'),
padding.OAEP(
mgf=padding.MGF1(algorithm=hashes.SHA256()),
algorithm=hashes.SHA256(),
label=None
)
)
return ciphertext
def decrypt_message(private_key, ciphertext):
"""Decrypts a ciphertext using the recipient's private key."""
plaintext = private_key.decrypt(
```

```
ciphertext,
padding.OAEP(
mgf=padding.MGF1(algorithm=hashes.SHA256()),
algorithm=hashes.SHA256(),
label=None
)
)
return plaintext.decode('utf-8')

if __name__ == "__main__":
# User 1 generates their keys
print("User 1: Generating RSA key pair...")
user1_private_key, user1_public_key = generate_rsa_key_pair()
print("User 1: Key pair generated.")

# User 2 generates their keys
print("\nUser 2: Generating RSA key pair...")
user2_private_key, user2_public_key = generate_rsa_key_pair()
print("User 2: Key pair generated.")

# User 1 sends a message to User 2
original_message_user1 = "Hello User 2, this is a secret message from User 1!"
print(f"\nUser 1: Original message to User 2: '{original_message_user1}'")

# User 1 encrypts the message using User 2's public key
encrypted_message_user1_to_user2 = encrypt_message(user2_public_key, original_message_user1)
print(f"User 1: Encrypted message (ciphertext): {encrypted_message_user1_to_user2}")
# User 2 receives and decrypts the message using their private key
decrypted_message_user2 = decrypt_message(user2_private_key,
encrypted_message_user1_to_user2)
print(f"User 2: Decrypted message: '{decrypted_message_user2}'")

# User 2 sends a reply to User 1
```

```
original_message_user2 = "Hi User 1, I received your message securely!"
print(f"\nUser 2: Original message to User 1: '{original_message_user2}'")

# User 2 encrypts the reply using User 1's public key
encrypted_message_user2_to_user1 = encrypt_message(user1_public_key, original_message_user2)
print(f"User 2: Encrypted reply (ciphertext): {encrypted_message_user2_to_user1}")

# User 1 receives and decrypts the reply using their private key
decrypted_message_user1 = decrypt_message(user1_private_key,
encrypted_message_user2_to_user1)
print(f"User 1: Decrypted reply: '{decrypted_message_user1}'")
```

Output:

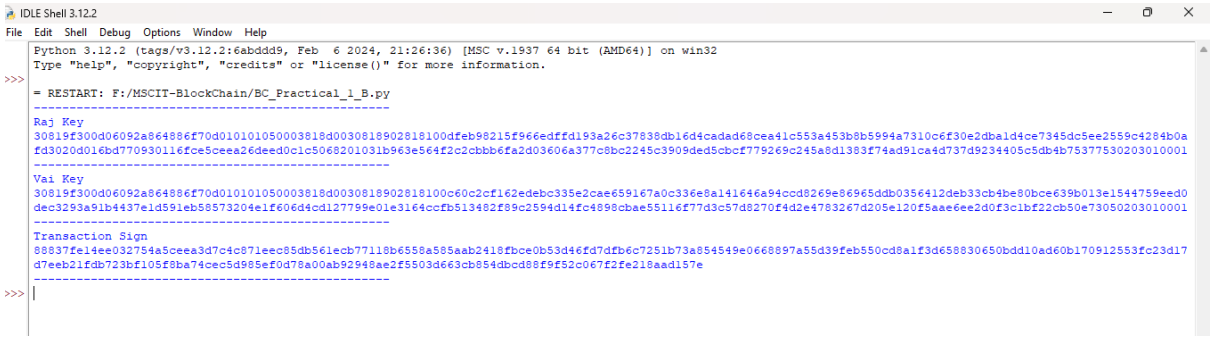
B. Allow users to create multiple transactions and display them in an organised format.

Code

```
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')
Raj = Client()
print("-"*50)
print("Raj Key")
print(Raj.identity)
Vai = Client()
print("-"*50)
print("Vai Key")
print(Vai.identity)
t = Transaction(Raj, Vai.identity, 10.0)
print("-"*50)
print("Transaction Sign")
signature = t.sign_transaction()
print(signature)
print("-"*50)
```

Output:



C. Create a Python class named Transaction with attributes for sender, receiver, and amount. Implement a method within the class to transfer money from the sender's account to the receiver's account.

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()
```

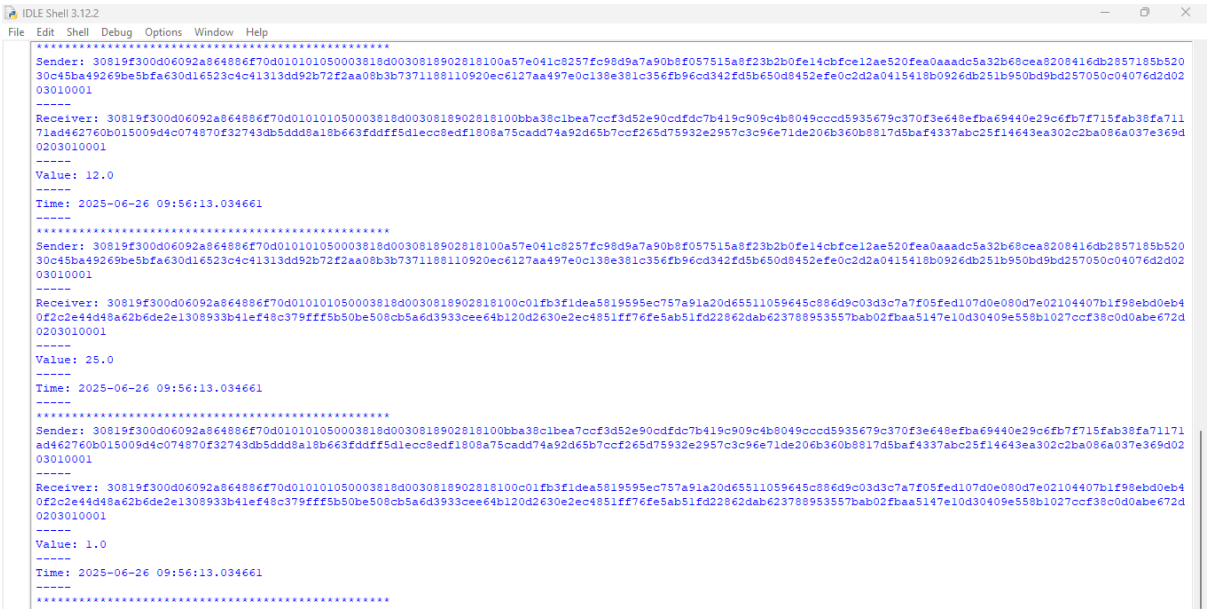
```
def to_dict(self):
if self.sender == "Genesis":
identity = "Genesis"
else:
identity = self.sender.identity
return collections.OrderedDict(
{
"sender": identity,
"receiver": self.receiver,
"value": self.value,
"time": self.time,
}
)
def sign_transaction(self):
private_key = self.sender._private_key
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:

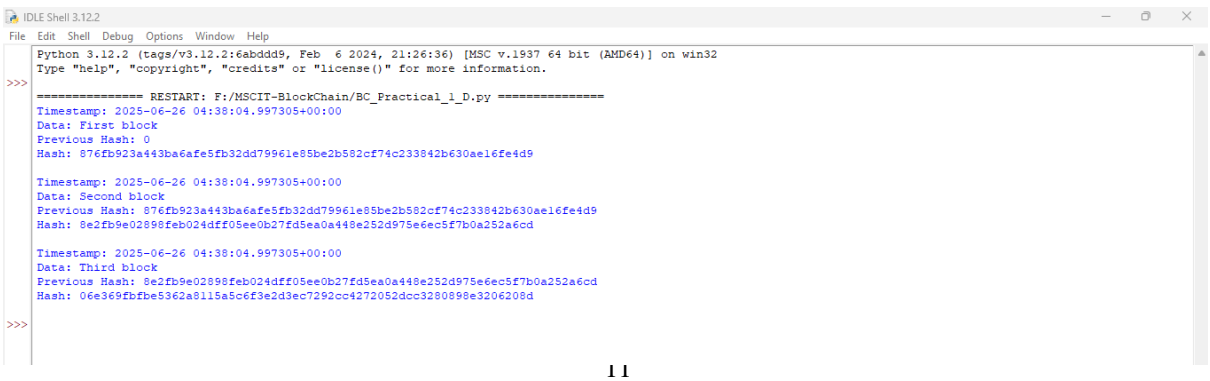


D. Implement a function to add new blocks to the miner and dump the blockchain

Code:

```
import datetime

import hashlib

# Create a class with two functions

class Block:

def __init__(self, data, previous_hash):

self.timestamp = datetime.datetime.now(datetime.timezone.utc)

self.data = data

self.previous_hash = previous_hash

self.hash = self.calc_hash()

def calc_hash(self):

sha = hashlib.sha256()

hash_str = self.data.encode("utf-8")

sha.update(hash_str)

return sha.hexdigest()

if __name__ == "__main__":

# 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")
```

Output:



Practical No. 2

A. Write a python program to demonstrate mining

Code:

```
//npm install web3
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 etheron Ganache with coinbase address:${coinbaseacc1}');
while(true)
{ try{
await web3.eth.sendTransaction({
from:coinbaseacc1, to:coinbaseacc2,
value:50,
});
console.log('Hii Shiva Mined a new block!');
}catch(err){ console.error(err);
} }
} mine();
```

Output:

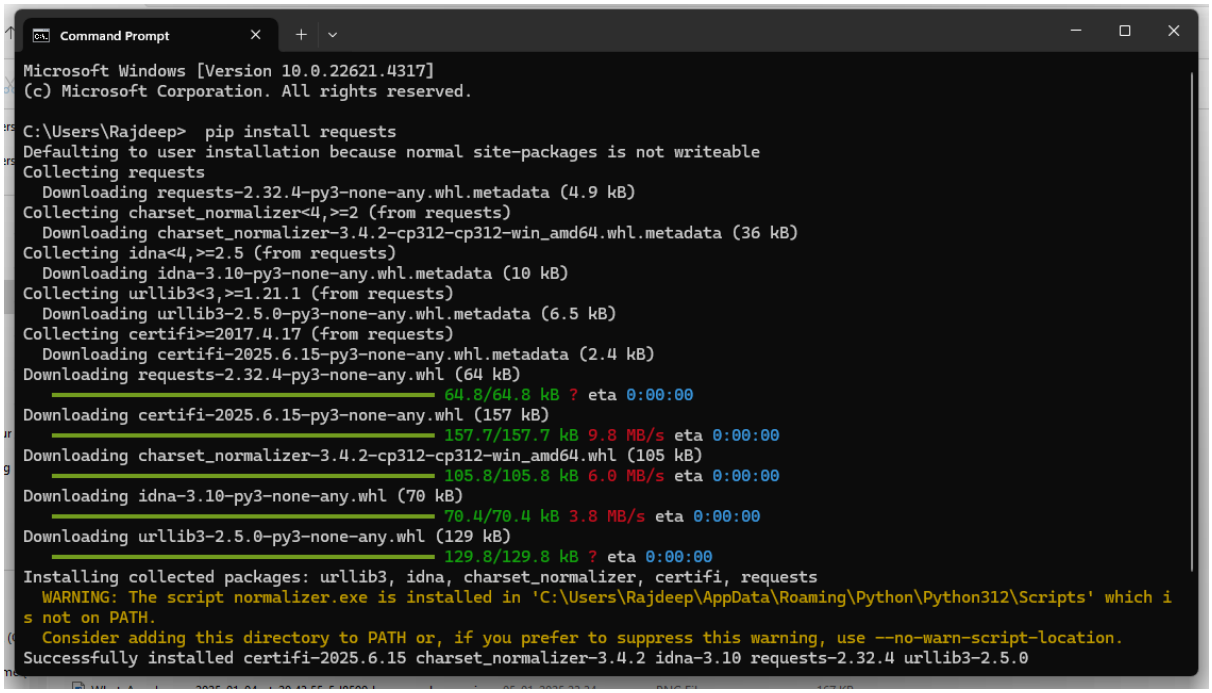
```
C:\Users\schau\Music\Blockchain Practicals>node ethermine.js
Mining etheron Ganache with coinbase address:${coinbaseacc1}
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
Mined a new block!
```

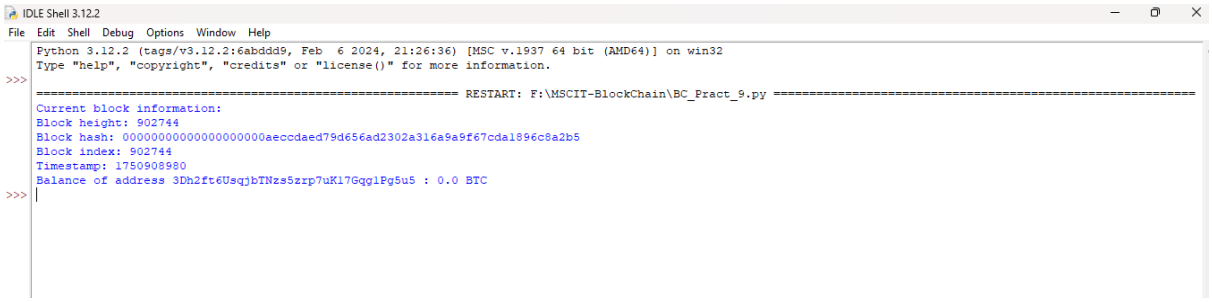
B. Demonstrate the use of the Bitcoin Core API to interact with a Bitcoin Core node.

Bitcoin Core Api

```
# pip install requests
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:


Command Prompt
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Rajdeep> pip install requests
Defaulting to user installation because normal site-packages is not writeable
Collecting requests
Downloading requests-2.32.4-py3-none-any.whl.metadata (4.9 kB)
Collecting charset_normalizer<4,>=2 (from requests)
Downloading charset_normalizer-3.4.2-cp312-cp312-win_amd64.whl.metadata (36 kB)
Collecting idna<4,>=2.5 (from requests)
Downloading idna-3.10-py3-none-any.whl.metadata (10 kB)
Collecting urllib3<3,>=1.21.1 (from requests)
Downloading urllib3-2.5.0-py3-none-any.whl.metadata (6.5 kB)
Collecting certifi>=2017.4.17 (from requests)
Downloading certifi-2025.6.15-py3-none-any.whl.metadata (2.4 kB)
Downloading requests-2.32.4-py3-none-any.whl (64 kB)
64.8/64.8 kB ? eta 0:00:00
Downloading certifi-2025.6.15-py3-none-any.whl (157 kB)
157.7/157.7 kB 9.8 MB/s eta 0:00:00
Downloading charset_normalizer-3.4.2-cp312-cp312-win_amd64.whl (105 kB)
105.8/105.8 kB 6.0 MB/s eta 0:00:00
Downloading idna-3.10-py3-none-any.whl (70 kB)
70.4/70.4 kB 3.8 MB/s eta 0:00:00
Downloading urllib3-2.5.0-py3-none-any.whl (129 kB)
129.8/129.8 kB ? eta 0:00:00
Installing collected packages: urllib3, idna, charset_normalizer, certifi, requests
WARNING: The script normalizer.exe is installed in 'C:\Users\Rajdeep\AppData\Roaming\Python\Python312\Scripts' which i
s not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed certifi-2025.6.15 charset_normalizer-3.4.2 idna-3.10 requests-2.32.4 urllib3-2.5.0


IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: F:\MSCIT-BlockChain\BC_Pract_9.py ==========
Current block information:
Block height: 902744
Block hash: 00000000000000000000aeccdaed79d656ad2302a316a9a9f67cda1896c8a2b5
Block index: 902744
Timestamp: 1750908980
Balance of address 3Dh2ft6UsqjbTNzs5zrp7uK17Gqg1Pg5u5 : 0.0 BTC
>>>

C. Demonstrating the process of running a blockchain node on your local machine.

Make Sure you have Installed node.js in their System.

Code:

```
// npm install crypto-js
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,"09/06/2024","GenesisBlock","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(leti=1;i<this.chain.length;i++){ constcurrentBlock = this.chain[i];
constpreviousBlock = this.chain[i-1];
if(currentBlock.hash != currentBlock.calculateHash()){ returnfalse;
}
if(currentBlock.previousHash != previousBlock.hash){ return false;
}
}
return true;
} }
//BlockchainImplementation
```

```
let myCoin=new Blockchain();
myCoin.addBlock(new Block(1,"09/06/2024",{amount:4}));
myCoin.addBlock(new Block(2,"09/06/2024",{amount:8}));
// console.log('Isblockchainvalid?'+myCoin.isChainValid());
console.log(JSON.stringify(myCoin,null,4))
```

Output:

```
C:\Users\schau\Music\Blockchain Practicals>node main.js
{
    "chain": [
        {
            "index": 0,
            "timestamp": "09/06/2024",
            "data": "GenesisBlock",
            "previousHash": "0",
            "hash": "aa9262d28a2dd660edee1f21c813d95cfb5ef420da4776b2f1ad2454d1848895"
        },
        {
            "index": 1,
            "timestamp": "09/06/2024",
            "data": {
                "amount": 4
            },
            "previousHash": "aa9262d28a2dd660edee1f21c813d95cfb5ef420da4776b2f1ad2454d1848895",
            "hash": "05a1db13df9faa0e3d2e845e177538e310a68c32d6edcb45740fedcfabe1db54"
        },
        {
            "index": 2,
            "timestamp": "09/06/2024",
            "data": {
                "amount": 8
            },
            "previousHash": "05a1db13df9faa0e3d2e845e177538e310a68c32d6edcb45740fedcfabe1db54",
            "hash": "57f583ebe09d59c17d73892ec244180afa91d0e23afdcc853c2f338ca267d7af"
        }
    ]
}
```

D. Demonstrate mining using geth on your private network.

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

Genesis.json

```
{
"config": {
"chainId": 987, "homesteadBlock": 0,
"eip150Block": 0, "eip155Block": 0, "eip158Block": 0
},
"difficulty": "0x400",
"gasLimit": "0x8000000",
"alloc": {}
}
```

CustomGenesis.json

```
{
    "config": {
        "chainId": 987,
        "homesteadBlock": 0,
        "eip150Block": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty": "0x400",
    "gasLimit": "0x8000000",
    "alloc": {}
}
```

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

testnet-blockchain.

```
C:\Users\Achsah>geth account new --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical
\ethermine
INFO [04-20|20:03:09.337] Maximum peer count                       ETH=50 LES=0 total=50
Your new account is locked with a password. Please give a password. Do not forget this password.
Password:
Repeat password:

Your new key was generated

Public address of the key:   0x77CB2BdBC0f1743bC73E92f1a8b1AB80BEDB35AE
Path of the secret key file: C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine\key
store\UTC--2023-04-20T14-33-26.959134300Z--77cb2bdbc0f1743bc73e92f1a8b1ab80bedb35ae

- You can share your public address with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your funds!
- You must BACKUP your key file! Without the key, it's impossible to access account funds!
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!
```

Step 3-> Run command geth account new –datadir
C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine

```
C:\Users\Achsah>geth --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine i
nit C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine\genesis.json
Fatal: invalid genesis file: math/big: cannot unmarshal "\"3792\"" into a *big.Int

C:\Users\Achsah>geth --datadir C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine i
nit C:\Users\Achsah\Documents\MScIT\sem4\blockchain_practical\ethermine\genesis.json
INFO [04-20|20:23:47.707] Maximum peer count                       ETH=50 LES=0 total=50
INFO [04-20|20:23:47.717] Set global gas cap                       cap=50,000,000
INFO [04-20|20:23:47.720] Using leveldb as the backing database
INFO [04-20|20:23:47.720] Allocated cache and file handles         database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata cache=16.00MiB handles=16
INFO [04-20|20:23:47.741] Using LevelDB as the backing database
INFO [04-20|20:23:47.765] Opened ancient database                  database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata\ancient/chain readonly=false
INFO [04-20|20:23:47.767] Writing custom genesis block
INFO [04-20|20:23:47.773] Persisted trie from memory database      nodes=1 size=147.00B time="636.4µ
```

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

```
C:\Users\Achsah>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
INFO [04-20|20:29:41.383] Maximum peer count                       ETH=50 LES=0 total=50
INFO [04-20|20:29:41.389] Set global gas cap                       cap=50,000,000
INFO [04-20|20:29:41.392] Allocated trie memory caches             clean=154.00MiB dirty=256.00MiB
INFO [04-20|20:29:41.396] Using leveldb as the backing database
INFO [04-20|20:29:41.396] Allocated cache and file handles         database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata cache=512.00MiB handles=8192
INFO [04-20|20:29:41.412] Using LevelDB as the backing database
INFO [04-20|20:29:41.420] Opened ancient database                  database=C:\Users\Achsah\Document
s\MScIT\sem4\blockchain_practical\ethermine\geth\chaindata\ancient/chain readonly=false
INFO [04-20|20:29:41.423] Disk storage enabled for ethash caches   dir=C:\Users\Achsah\Documents\MSc
IT\sem4\blockchain_practical\ethermine\geth\ethash count=3
INFO [04-20|20:29:41.424] Disk storage enabled for ethash DAGs     dir=C:\Users\Achsah\AppData\Local
\Ethash count=2
INFO [04-20|20:29:41.426] Initialising Ethereum protocol           network=5777 dbversion=<nil>
INFO [04-20|20:29:41.427]
INFO [04-20|20:29:41.430] ------------------------------------------------------------------------
```

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

```
To exit, press ctrl-d or type exit
> INFO [04-20|20:29:45.021] Mapped network port                    proto=tcp extport=30303 intport=3030
NP IGDv1-IP1"

>
> miner.setEtherbase('0xC050FE4d9bAc591d29538e2FD9cCA848B29489D0')
true
> miner.start()
INFO [04-20|20:34:45.673] Updated mining threads                 threads=4
INFO [04-20|20:34:45.674] Transaction pool price threshold updated price=1,000,000,000
null
> INFO [04-20|20:34:45.683] Commit new sealing work                number=1 sealhash=2e6f57..6db9c6 und
=0 fees=0 elapsed=7.571ms
INFO [04-20|20:34:45.686] Commit new sealing work                number=1 sealhash=2e6f57..6db9c6 uncle
 fees=0 elapsed=9.940ms
INFO [04-20|20:34:47.975] Generating DAG in progress             epoch=0 percentage=0 elapsed=1.636s
INFO [04-20|20:34:49.873] Generating DAG in progress             epoch=0 percentage=1 elapsed=3.534s
```

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

```
INFO [04-20|20:38:42.556] Generating DAG in progress               epoch=0 percentage=98 elapsed=3m5
6.216s
INFO [04-20|20:38:46.897] Generating DAG in progress               epoch=0 percentage=99 elapsed=4m0
.557s
INFO [04-20|20:38:46.901] Generated ethash verification cache      epoch=0 elapsed=4m0.561s
INFO [04-20|20:38:48.755] Successfully sealed new block            number=1 sealhash=2e6f57..6db9c6
hash=ccf3e9..10adff elapsed=4m3.071s
INFO [04-20|20:38:48.765] "🔨 mined potential block"                number=1 hash=ccf3e9..10adff
INFO [04-20|20:38:48.756] Commit new sealing work                  number=2 sealhash=cb4ba0..84eldd
uncles=0 txs=0 gas=0 fees=0 elapsed="504.9µs"
INFO [04-20|20:38:48.770] Commit new sealing work                  number=2 sealhash=cb4ba0..84eldd
uncles=0 txs=0 gas=0 fees=0 elapsed=14.488ms
INFO [04-20|20:38:49.389] Successfully sealed new block            number=2 sealhash=cb4ba0..84eldd
hash=4c7137..a04b67 elapsed=632.526ms
```

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

```
INFO [04-20|20:39:21.980] Commit new sealing work                  number=17 sealhash=923697..cb5b4d
 uncles=0 txs=0 gas=0 fees=0 elapsed=117.201ms
INFO [04-20|20:39:21.984] Ethereum protocol stopped
INFO [04-20|20:39:22.046] Transaction pool stopped
INFO [04-20|20:39:22.047] Writing cached state to disk             block=16 hash=f09f60..c23237 root
=0c083a..cddeff
INFO [04-20|20:39:22.081] Persisted trie from memory database      nodes=3 size=408.00B time=1.5741m
s gcnodes=0 gcsize=0.00B gctime=0s livenodes=31 livesize=3.83KiB
INFO [04-20|20:39:22.087] Writing cached state to disk             block=15 hash=d73b6d..f4a2cf root
=903c8d..6038c0
INFO [04-20|20:39:22.089] Persisted trie from memory database      nodes=2 size=262.00B time=0s
  gcnodes=0 gcsize=0.00B gctime=0s livenodes=29 livesize=3.58KiB
INFO [04-20|20:39:22.098] Writing snapshot state to disk           root=d56154..abe42a
INFO [04-20|20:39:22.130] Persisted trie from memory database      nodes=0 size=0.00B   time=0s
  gcnodes=0 gcsize=0.00B gctime=0s livenodes=29 livesize=3.58KiB
INFO [04-20|20:39:22.135] Writing clean trie cache to disk         path=C:\Users\Achsah\Documents\MS
cIT\sem4\blockchain_practical\ethermine\geth\triecache threads=4
INFO [04-20|20:39:22.323] Persisted the clean trie cache           path=C:\Users\Achsah\Documents\MS
cIT\sem4\blockchain_practical\ethermine\geth\triecache elapsed=143.729ms
INFO [04-20|20:39:22.490] Blockchain stopped
```

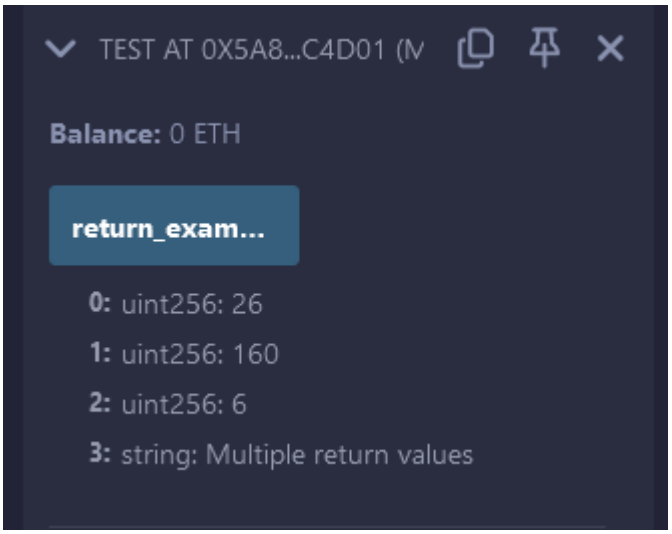
Practical No. 3

A. Write a Solidity program that demonstrates various types of functions including regular functions, view functions, pure functions, and the fallback function.

1. Functions

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

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);
   }

   }
```

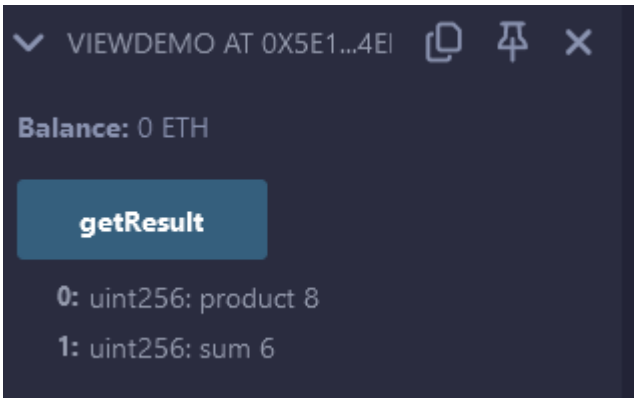
Output:



2. View Function

```
pragma solidity ^0.5.0;

contract ViewDemo
   { uint256 num1 =
   2; uint256 num2
   = 4;

   function getResult() public view returns (uint256 product, uint256 sum) {
      product = num1 * num2;
      sum = num1 + num2;
   }
}
```

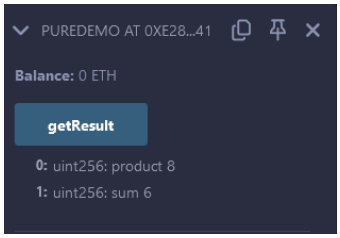
Output:



3. Pure Function:

```
pragma solidity ^0.5.0;

contract PureDemo {
   function getResult() public pure returns (uint256 product, uint256 sum) {
      uint256 num1 = 2;
      uint256 num2 = 4;
      product = num1 * num2;
      sum = num1 + num2;
   }
}
```

Output:

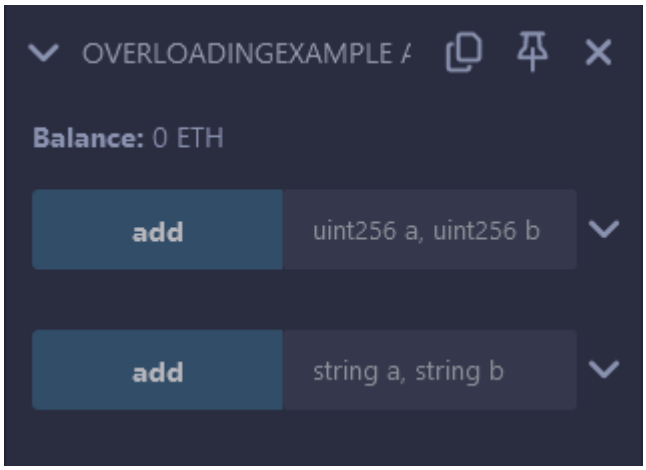


B. Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.

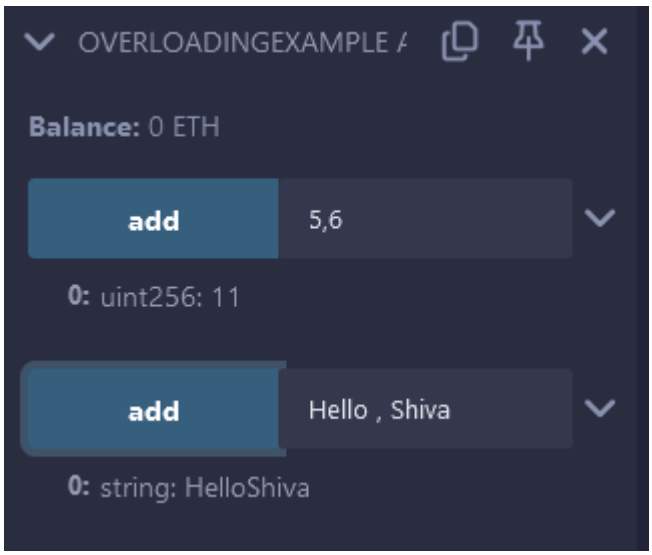
1. Function Overloading

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

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));
    }
}
```

Output:



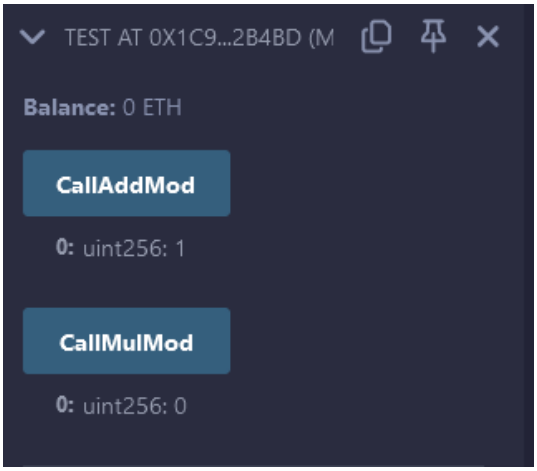
Give integer and string values to both add functions as below.



2. Mathematical Function

```
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);
   }
}
```

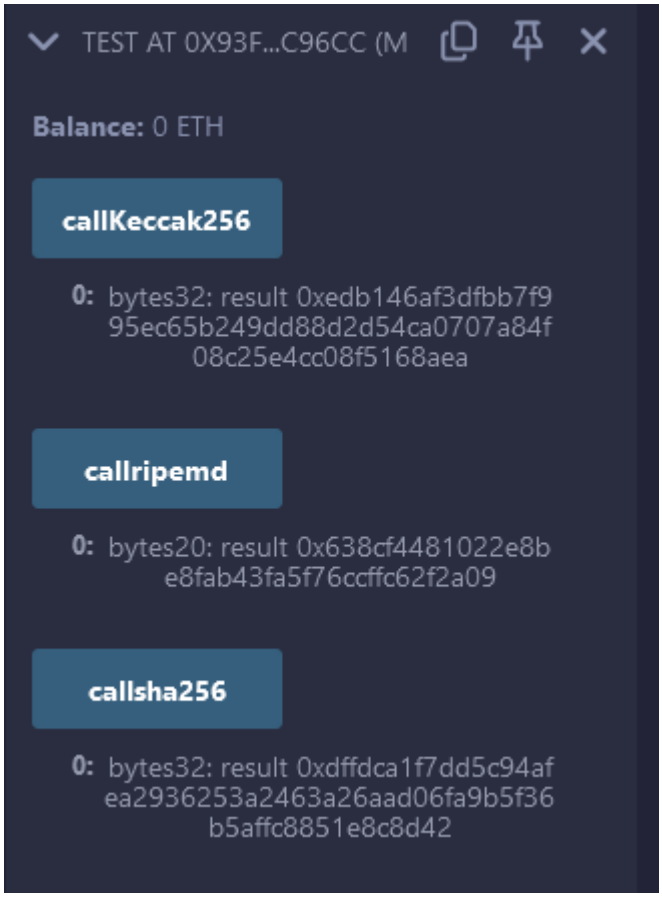
Output:



3. Cryptographic Functions.

```
pragma solidity ^0.5.0;

contract Test {
   function callKeccak256() public pure returns (bytes32 result) {
      return keccak256(abi.encodePacked("BLOCKCHAIN"));
   }

   function callsha256() public pure returns (bytes32 result) {
      return sha256(abi.encodePacked("BLOCKCHAIN"));
   }

   function callripemd() public pure returns (bytes20 result) {
      return ripemd160(abi.encodePacked("BLOCKCHAIN"));
   }
}
```

Output:

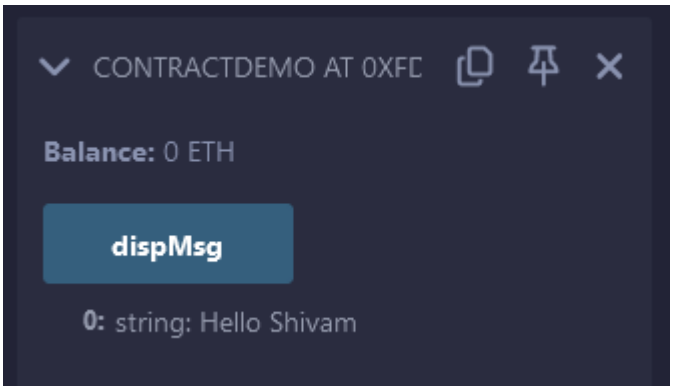

TEST AT 0X93F...C96CC (M
Balance: 0 ETH
callKeccak256
0: bytes32: result 0xedb146af3dfbb7f9 95ec65b249dd88d2d54ca0707a84f 08c25e4cc08f5168aea
callripemd
0: bytes20: result 0x638cf4481022e8b e8fab43fa5f76ccffc62f2a09
callsha256
0: bytes32: result 0xdffdca1f7dd5c94af ea2936253a2463a26aad06fa9b5f36 b5affc8851e8c8d42

C. Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.

1. Contracts

```
pragma solidity ^0.5.0;

contract ContractDemo {
   string message = "Hello Shivam";

   function dispMsg() public view returns (string memory) {
      return message;
   }
}
```

Output:

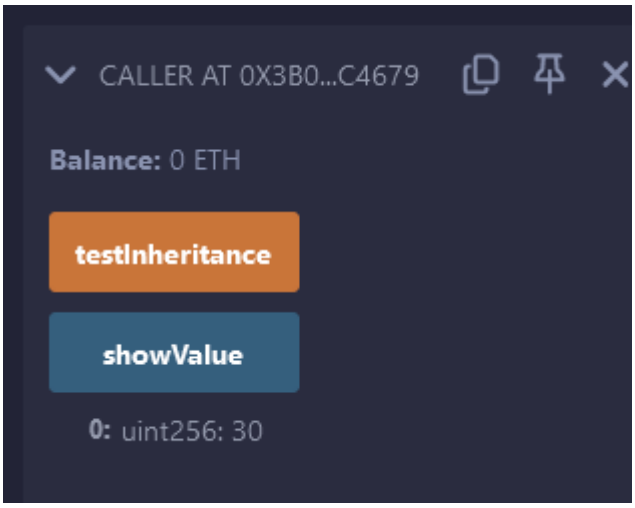


2. Inheritance

```
pragma solidity >=0.4.22 <0.6.0;

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 showValue() public view returns (uint256) {
      return cc.getValue();
   }
}
```

Output:

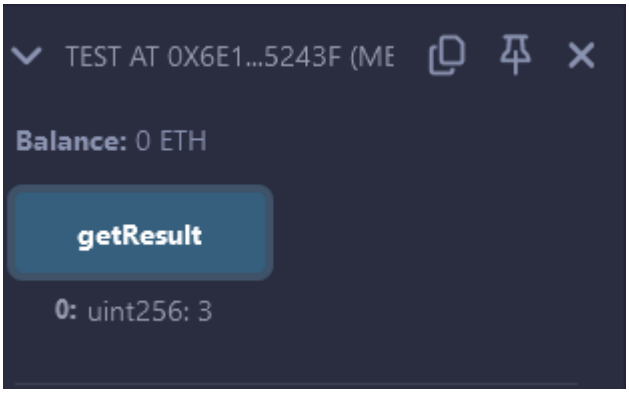


3. **Abstract Contracts**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.17;

contract Calculator {
   function getResult() external view returns (uint256);
}

contract Test is Calculator {
   constructor() public {}

   function getResult() external view returns (uint256) {
      uint256 a = 1;
      uint256 b = 2;
      uint256 result = a + b;
      return result;
   }
}
```

Output:

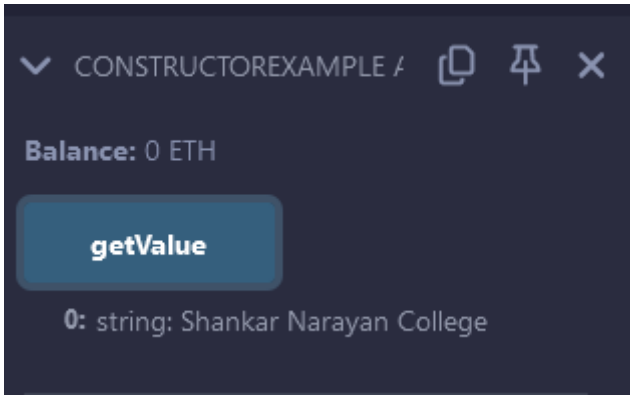
Output:



4. Constructor

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.0;

// Creating a contract
contract constructorExample {
   string str;

   constructor() public {
      str = "Shankar Narayan College";
   }

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

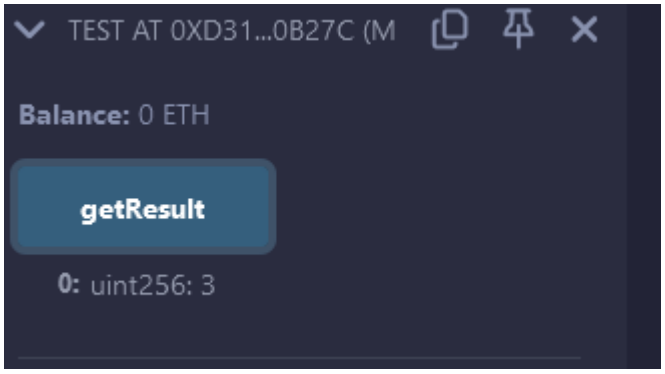
Output



5. Interfaces:

```
pragma solidity ^0.5.0;

interface Calculator {
   function getResult() external view returns (uint);
}

contract Test is Calculator {
   constructor() public {}

   function getResult() external view returns (uint) {
      uint a = 1;
      uint b = 2;
      uint result = a + b;
      return result;
   }
}
```

Output:



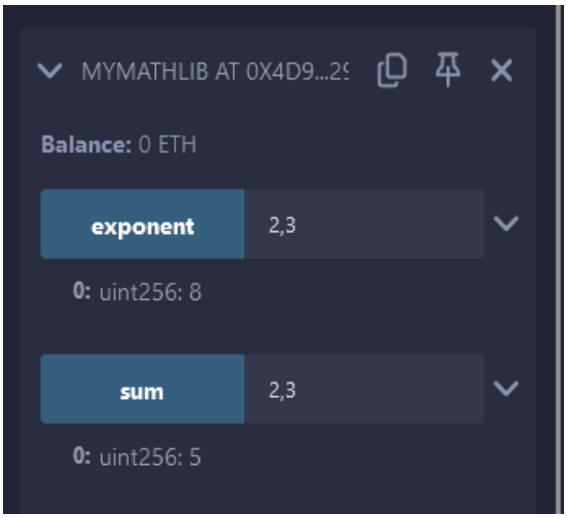
D. Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling.

1. Libraries

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;

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;
    }
}
```

```
/ SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;

import "contracts/myLIB.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);
    }
```

Output



2. Assembly

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

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)
      }
   }
}
```

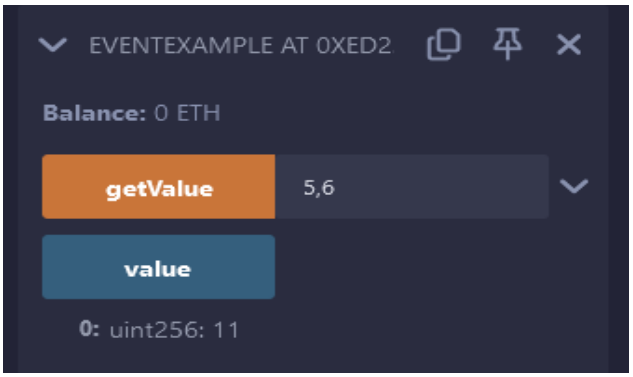
Output

3. Events

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.0;

// Creating a contract
contract eventExample {
   // Declaring state variables
   uint256 public value = 0;
   // Declaring an event
   event Increment(address owner);
```

```
    // Defining a function for logging event
    function getValue(uint256 _a, uint256 _b) public {
        emit Increment(msg.sender);  // Emitting the Increment event with the caller's address
        value = _a + _b;  // Updating the value state variable
    }
}
```

Output:



4. **Error Handling**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.17;

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;
    }
}
```

Output: