Practical 1

Write the following programs for Blockchain in Python:

Aim : (I) A simple client class that generates the private and public keys by using the built in Python RSA algorithm and tests it .

Input:

```
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
pip install pycryptodome
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

import binascii
class Client:
 def __init__(self):
     random = Crypto.Random.new().read
     self._private_key = RSA.generate(1024, random)
     self._public_key = self._private_key.publickey()
     self._signer = PKCS1_v1_5.new(self._private_key)
```

Output:

In [1]:
```
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
```

In [2]:
```
pip install pycryptodome
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting pycryptodome
  Downloading pycryptodome-3.22.0-cp37-abi3-win_amd64.whl (1.8 MB)
     ---------------------------------------- 1.8/1.8 MB 14.3 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.22.0
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
```

In [3]:
```
import binascii
class Client:
 def __init__(self):
      random = Crypto.Random.new().read
      self._private_key = RSA.generate(1024, random)
      self._public_key = self._private_key.publickey()
      self._signer = PKCS1_v1_5.new(self._private_key)
```

Aim: (II) A transaction class to send and receive money and test it .

Input:

```
import datetime
import collections
import binascii
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA

class Client:
    def __init__(self):
        # Generate a public/private key pair
        key = RSA.generate(2048)
        self._private_key = key
        self._public_key = key.publickey()

    def identity(self):
        # Return the public key in hexadecimal format
        return binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')

class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()

    def to_dict(self):
        if self.sender == "Genesis":
            identity = "Genesis"
        else:
            identity = self.sender.identity()

        return collections.OrderedDict({
            'sender': identity,
            'recipient': self.recipient,
            'value': self.value,
            'time': self.time
        })

    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')
```

# Example usage
Dinesh = Client()
Ramesh = Client()

t = Transaction(Dinesh, Ramesh, 5.0)
signature = t.sign_transaction()
print(signature)

output:
-

```
        # Generate a public/private key pair
        key = RSA.generate(2048)
        self._private_key = key
        self._public_key = key.publickey()

    def identity(self):
        # Return the public key in hexadecimal format
        return binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')

class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()

    def to_dict(self):
        if self.sender == "Genesis":
            identity = "Genesis"
        else:
            identity = self.sender.identity()

        return collections.OrderedDict({
            'sender': identity,
            'recipient': self.recipient,
            'value': self.value,
            'time': self.time
        })

    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')


# Example usage
Dinesh = Client()
Ramesh = Client()

t = Transaction(Dinesh, Ramesh, 5.0)
signature = t.sign_transaction()
print(signature)
```

```
2dc3159c0d2cb5e5c234c62b19916b3e1c26721ad1486dff617e1ff54e350400b10a749b322f9d03fbda823c5ff1ad4a04c2c6fd6a06e0b34f49e29aa3aa59a
137908ec12abb154fa640f068dc42a22795561270 6adbc027442a240ac6f10842752b4239ce4390f04b7564f24c72ed5e324dc047b5db61cd7d38dd2be28b6e
4a855246381f0b673161df7c37338218de0ef268b0e82f8b2944a1e18fd7f60cd92ea93bc270a90a71ffdd075c5a2df0935fb26edc3d62c03810a609dc98f81
3255d81c3d59fe0317cc9ac88eb85b00d6abe5ce0a0f2563b736c8905ab650ecdcda4c2fac36dd1a3dd10356d872f83d884af45470edf142948bdd1bdd937b2
1e88
```

Practical 2

Write the following programs for Blockchain in Python :

Aim:(I).Create multiple transactions and display them.

```
# Function to display the transaction details
def display_transaction(transaction):
    transaction_dict = transaction.to_dict()
    print(f"Sender: {transaction_dict['sender']}")
    print(f"Recipient: {transaction_dict['recipient']}")
    print(f"Value: {transaction_dict['value']}")
    print(f"Time: {transaction_dict['time']}")
    print(f"Signature: {transaction.sign_transaction()}")
    print("-----")



# Main usage example
transactions = []

Dinesh = Client()
Ramesh = Client()
Seema = Client()
Vijay = Client()

# Creating and signing transactions
t1 = Transaction(Dinesh, Ramesh.identity(), 15.0)
t1.sign_transaction()
transactions.append(t1)

t2 = Transaction(Dinesh, Seema.identity(), 6.0)
t2.sign_transaction()
transactions.append(t2)

t3 = Transaction(Ramesh, Vijay.identity(), 2.0)
t3.sign_transaction()
transactions.append(t3)

t4 = Transaction(Seema, Ramesh.identity(), 4.0)
t4.sign_transaction()
transactions.append(t4)

t5 = Transaction(Vijay, Seema.identity(), 7.0)
t5.sign_transaction()
transactions.append(t5)

t6 = Transaction(Ramesh, Seema.identity(), 3.0)
t6.sign_transaction()
transactions.append(t6)
```

```
t7 = Transaction(Seema, Dinesh.identity(), 8.0)
t7.sign_transaction()
transactions.append(t7)

t8 = Transaction(Seema, Ramesh.identity(), 1.0)
t8.sign_transaction()
transactions.append(t8)

t9 = Transaction(Vijay, Dinesh.identity(), 5.0)
t9.sign_transaction()
transactions.append(t9)

t10 = Transaction(Vijay, Ramesh.identity(), 3.0)
t10.sign_transaction()
transactions.append(t10)

# Display all transactions
for transaction in transactions:
    display_transaction(transaction)
    print('----------')
```

```
--------------
Sender: 30820122300d06092a864886f70d01010105000382010f00308201 0a0282010100d49e9804d0c51b860429f10ef
81b753e036dc1787b837991b5f4fa8262741abee7b6ff4a3a7047ac6899f51ee803862c422dd2e6171587325d62104a97f6
fc4aa62665ac8cde5f3c8186c866a26187cc9b465843e21b15fdd36b1985a536676fd0b5baeb806272a4a2576bcc012adad
33ef01fdc7f844d441e44a9cd72b06cc04504778fdb4ea34df604ebf765f0f2178b963aa8580925f70650767ad6f119e699
256ffbf4a412cffc89dc4e50b60699ec148666d13757d44b8ba658e7e313a6c61b8ac059b53377fe03bb4f4cc7f000637ed
d38d3f162268079c0309ba4eb6a91a3bb6963919a3d51c188e1784aef17688c4cfb44bd317fd75a1bb5420ee5d902030100
01
Recipient: 30820122300d06092a864886f70d01010105000382010f00308201 0a0282010100b73458fc47eafaf662adbf
60bc4dbf592498876b824d84e5a9151d84ff1b8748e22343772b2e1364ee1bdf65684815454719bc2d46697914b7f96f348
3026b9125405e793acb25f50eade8cbb754c14b3e7b67d093d04d79a170dcc9997d378c0666c671a7996a67a17ee1863d78
a4402a77d6da5f7aba5005e7e6fe1ebfb56bf4c547fda431f5ab7a53b1b4c9186cb6377fb6bc7c6387315b8e18e9a36e6c5
1c66c2682643f6a82fbefe4db3be00a08812e5344dd2a25b8c71ccafd10965554e595a903eaeffdd8950ba79dd8db91180a
e80ed890691dcbf89d4d014f4bd958f38e3b7e89d95ad1113cf06468f6b6829dcce6a6d50a2981f27e939fcdc8645502030
10001
Value: 2.0
Time: 2025-04-09 10:29:10.203779
Signature: 561d8e0cab2b581f9e8ded90716cbbc57d9ff3060d290eb69400f4440fd780e5a0d38539b17983f8817a34de
4b033e09dd2614eb4504e9ca8050d1205ae31046de05e2e06fd8618b3c13cb603cbe1fcaf3e1e3e606d8824becc1fab03d5
```

Aim:(II). Create a blockchain, a genesis block and execute it .

```
# Block and blockchain definitions
TPCoins = []

# Genesis block transaction
t0 = Transaction("Genesis", Dinesh.identity(), 500.0)
block0 = Block()

# Genesis block previous block hash is "0"
block0.previous_block_hash = "0"
block0.verified_transactions.append(t0)

# Calculate hash for the first block (Genesis block)
block0.Nonce = "0000"  # Example Nonce (can be set to an actual mined value later)
digest = block0.calculate_hash()
last_block_hash = digest

# Add the first block to the blockchain
TPCoins.append(block0)

# Function to dump the blockchain
def dump_blockchain(chain):
    print(f"Number of blocks in the chain: {len(chain)}")
    for x in range(len(chain)):
        block_temp = chain[x]
        print(f"Block # {x}")
        for transaction in block_temp.verified_transactions:
            display_transaction(transaction)
            print('---------')
        print('======================================')

# Dump the blockchain
dump_blockchain(TPCoins)
```

output:

```
Number of blocks in the chain: 1
Block # 0
Sender: Genesis
Recipient: 30820122300d06092a864886f70d0101010500038201 0f003082010a0282010100b7bf65a192a92cce44828bd0
a050364635af7426464595c217174ac4a1f91ab31432579c0df2163b59b718a6d70c31f8153da7ad986757d1737d7923a9ffe
22edc0c99c1d853f26775aeade705e3727286fcc4828416810d332dc7e9f143a18ef3fb310aea4b162c3f353cb9f6f034f01f
49517f251f93901130db16ee5aaa1d418b33cf78e0caa9b9222d1eade6e9af70a9da7d12d676b41673a3db7b12208f8df1f5e
1ddf934b9829a43c0e41a97c821d5b7500df6b286b2e054aaa51c409cae05c6e980b4a401a8e1d9699bafa056e5592a9cdd62
1efa204b850297ae588a333993ab65113ecdc8cebcb4678ec3d03d26410cc675d835587be25d893f66850203010001
Value: 500.0
Time: 2025-04-09 10:29:14.625778
Signature: Genesis transaction - no signature
-----
--------------
======================================
```

Practical 3

Write the following programs for Blockchain in Python :

Aim: (I). Create a mining function and test it .

(II). Add blocks to the miner and dump the blockchain .

```
import hashlib
import datetime

# Define the Transaction class
class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()

    def to_dict(self):
        return {
            'sender': self.sender,
            'recipient': self.recipient,
            'value': self.value,
            'time': self.time
        }

# Define the Block class
class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""
        self.timestamp = datetime.datetime.now()
        self.block_hash = ""

    def calculate_hash(self):
        # Concatenate the block's data (transactions, previous hash, nonce, timestamp) and hash
it
        block_data = str(self.verified_transactions) + str(self.previous_block_hash) +
str(self.Nonce) + str(self.timestamp)
        return hashlib.sha256(block_data.encode('utf-8')).hexdigest()

# Define the mining function
def sha256(message):
    return hashlib.sha256(message.encode('ascii')).hexdigest()

def mine(message, difficulty=1):
    assert difficulty >= 1
    prefix = '0' * difficulty  # '0' prefix to simulate proof of work
    for i in range(1000):  # Limit attempts to avoid infinite loop
```

```
        digest = sha256(str(message) + str(i))
        if digest.startswith(prefix):
            print(f"After {i} iterations, found nonce: {digest}")
            return digest  # Return the hash when found
    return None  # Return None if no valid hash is found

# Initialize variables for the blockchain and transactions
last_transaction_index = 0
last_block_hash = "0"  # Genesis block has no previous hash
TPCoins = []  # Blockchain (list of blocks)
transactions = [
    Transaction("Genesis", "Dinesh", 500),
    Transaction("Dinesh", "Ramesh", 15),
    Transaction("Dinesh", "Seema", 6),
    Transaction("Ramesh", "Vijay", 2),
    Transaction("Seema", "Ramesh", 4)
]

# Miner 1 adds a block
block = Block()
while last_transaction_index < len(transactions):  # Ensure not going out of range
    temp_transaction = transactions[last_transaction_index]
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine("Block 1", 2)  # Mine the block
block.block_hash = block.calculate_hash()  # Calculate the block's hash after mining

TPCoins.append(block)  # Add the mined block to the blockchain
last_block_hash = block.block_hash  # Update the last block's hash

# Miner 2 adds a block
block = Block()
while last_transaction_index < len(transactions):  # Ensure not going out of range
    temp_transaction = transactions[last_transaction_index]
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine("Block 2", 2)  # Mine the block
block.block_hash = block.calculate_hash()  # Calculate the block's hash after mining

TPCoins.append(block)  # Add the mined block to the blockchain
last_block_hash = block.block_hash  # Update the last block's hash

# Miner 3 adds a block
block = Block()
while last_transaction_index < len(transactions):  # Ensure not going out of range
    temp_transaction = transactions[last_transaction_index]
```

```
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine("Block 3", 2)  # Mine the block
block.block_hash = block.calculate_hash()  # Calculate the block's hash after mining

TPCoins.append(block)  # Add the mined block to the blockchain
last_block_hash = block.block_hash  # Update the last block's hash

# Function to display the blockchain
def dump_blockchain():
    print(f"Number of blocks in the chain: {len(TPCoins)}")
    for idx, block in enumerate(TPCoins):
        print(f"Block #{idx}")
        print(f"Block Hash: {block.block_hash}")
        for transaction in block.verified_transactions:
            print(transaction.to_dict())
        print("====================================")

# Display the entire blockchain
dump_blockchain()
```

output:

```
After 344 iterations, found nonce: 003d88c191521973a86a0cd2422d9c8661929c3dde67a4f13cf911ee760f5414
After 225 iterations, found nonce: 00d9f012f55b46919c6561a5b634f630cce11f9596052048fa29c720867a6ef8
After 16 iterations, found nonce: 004f3bf91979a243165bb6975d37c88c2f810580c03ef15109c10443d87ae7dd
Number of blocks in the chain: 3
Block #0
Block Hash: 7202135779d3ec55c9dec67e2aec549f14cf755ed0f5b0a12671b157093a9c0c
{'sender': 'Genesis', 'recipient': 'Dinesh', 'value': 500, 'time': datetime.datetime(2025, 4, 9, 10, 57, 20, 450471)}
{'sender': 'Dinesh', 'recipient': 'Ramesh', 'value': 15, 'time': datetime.datetime(2025, 4, 9, 10, 57, 20, 450471)}
{'sender': 'Dinesh', 'recipient': 'Seema', 'value': 6, 'time': datetime.datetime(2025, 4, 9, 10, 57, 20, 450471)}
{'sender': 'Ramesh', 'recipient': 'Vijay', 'value': 2, 'time': datetime.datetime(2025, 4, 9, 10, 57, 20, 450471)}
{'sender': 'Seema', 'recipient': 'Ramesh', 'value': 4, 'time': datetime.datetime(2025, 4, 9, 10, 57, 20, 450471)}
====================================
Block #1
Block Hash: 6ee8ef50cc671c3a9fd7806eb32e03510261db67e7c32576b5b771d0fb8a59a6
====================================
Block #2
Block Hash: fe3744532f87768b1130ff23d6b7c4d6346cb763be68eee3af03b2b90cdf23b8
====================================
```

Practical 4

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

(I)Variable

(II) Operators

(III) Decision Making

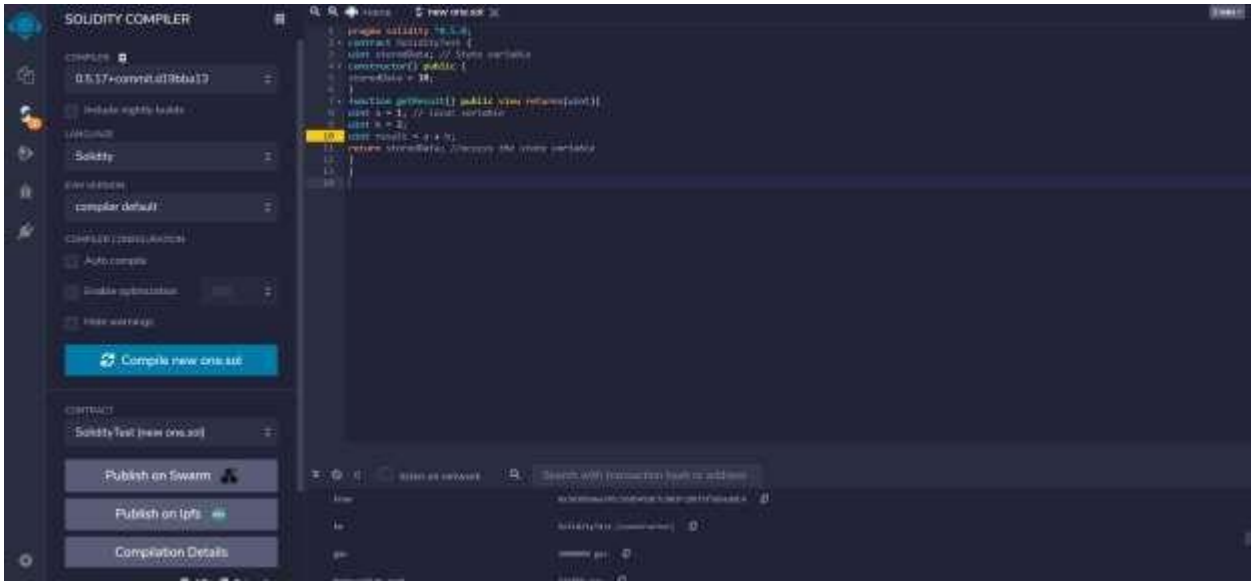
(IV) Strings

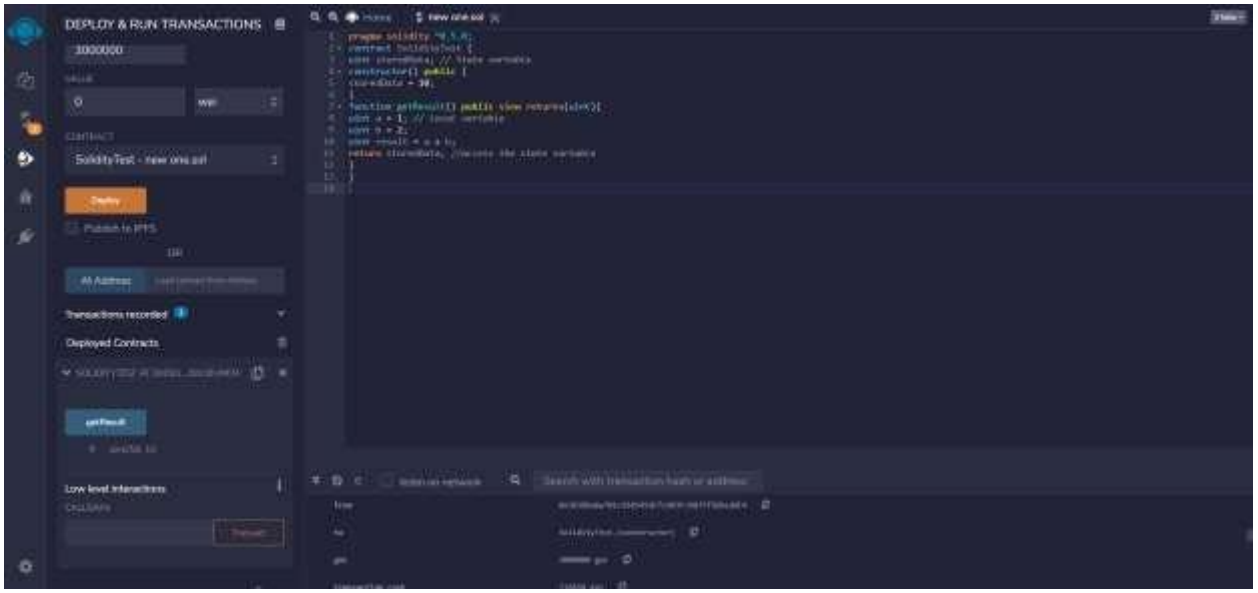
(V)Loops

(I). Variable

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

Output:

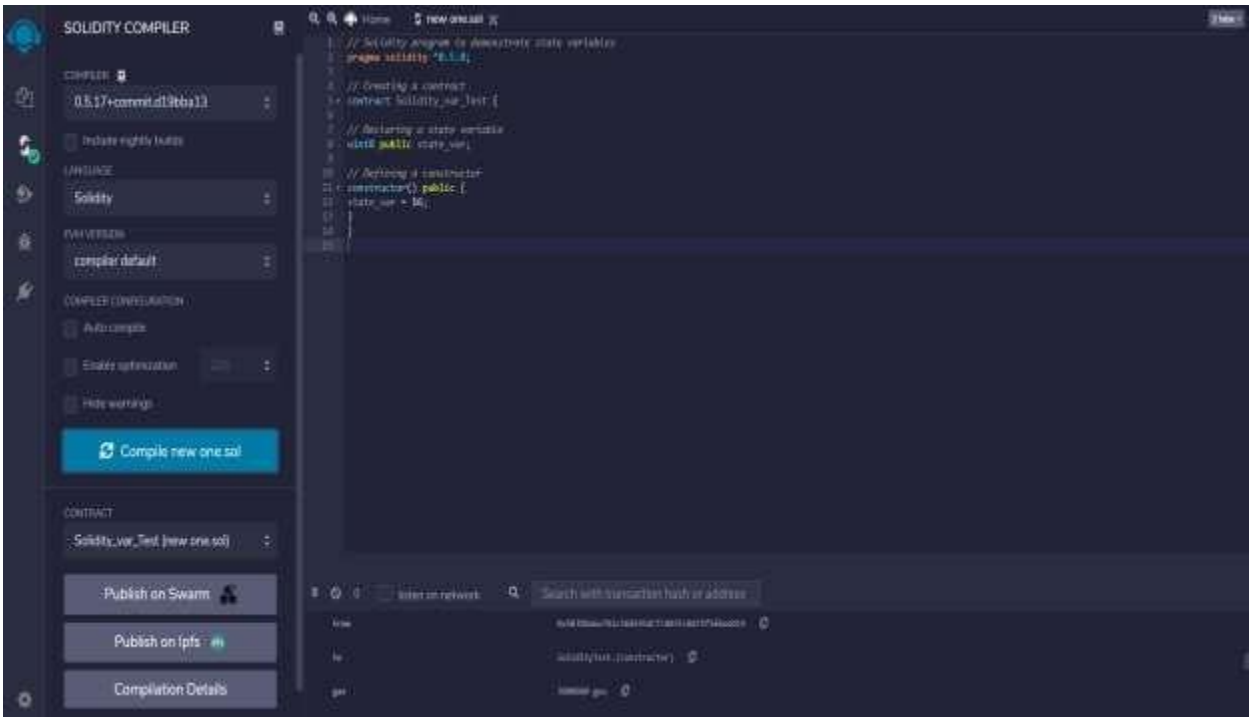

SOLIDITY COMPILER
Solidity
compiler default
Compile new one.sol
Publish on Swarm
Publish on Ipfs
Compilation Details

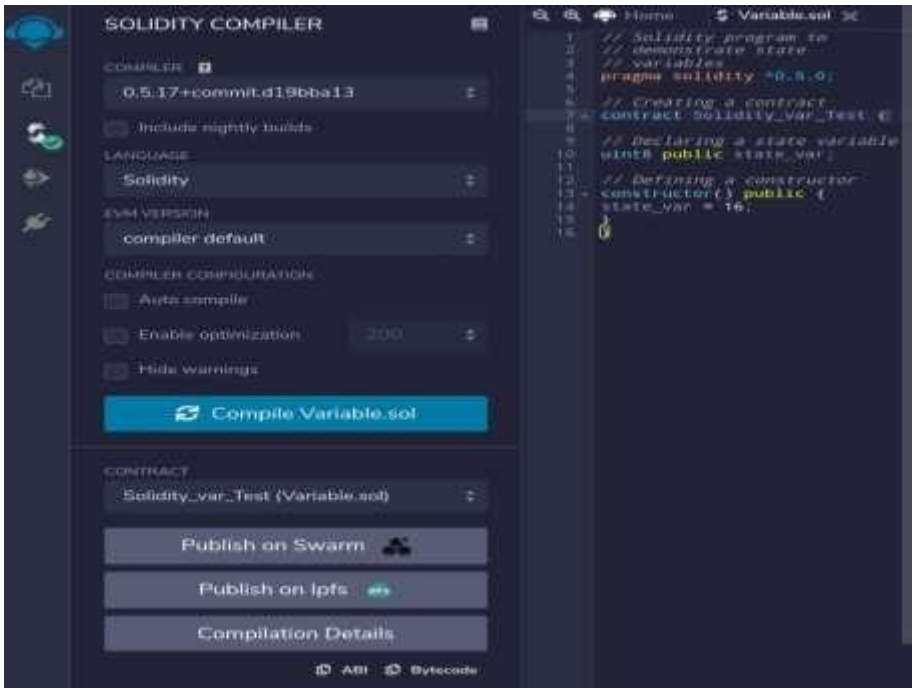

DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions

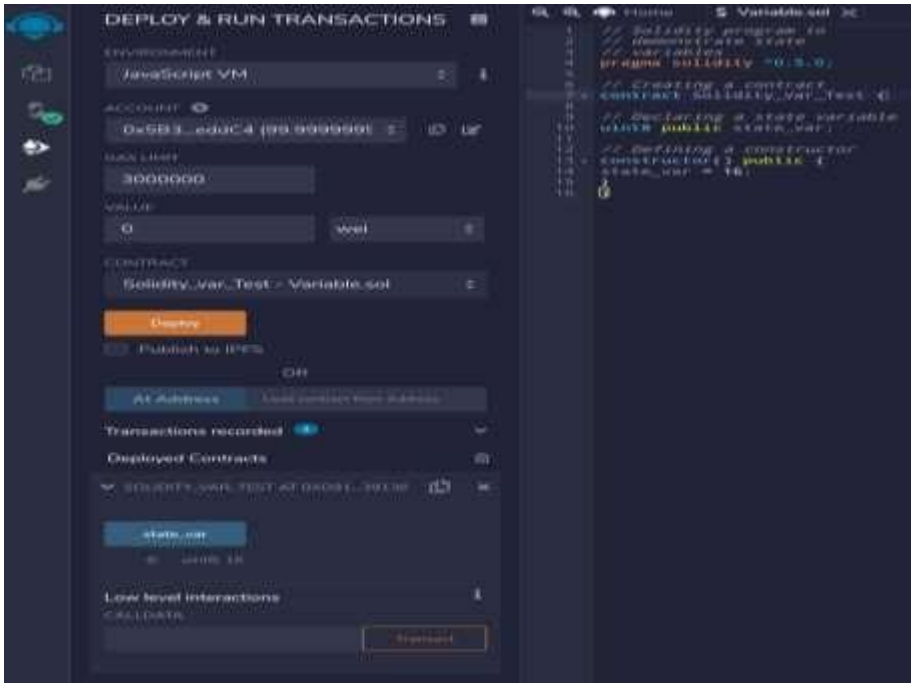
// **Solidity program to demonstrate state variables**

```
pragma solidity ^0.5.0;

// Creating a contract
contract Solidity_var_Test {

// Declaring a state variable
uint8 public state_var;

// Defining a constructor
constructor() public {
state_var = 16;
}
}
```

Output:



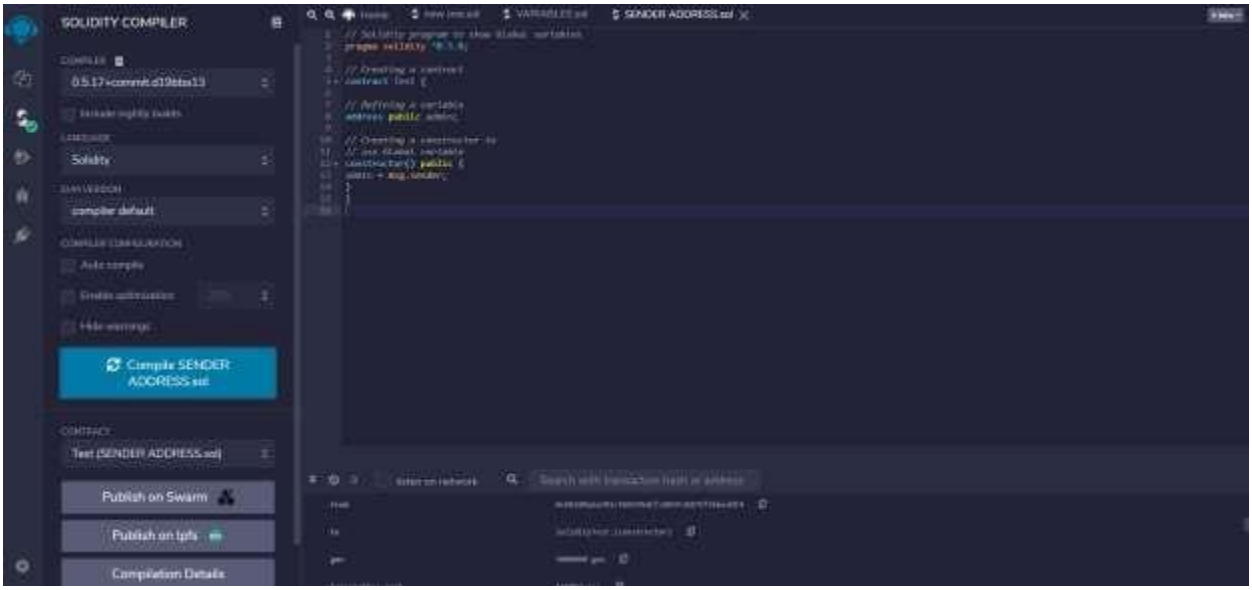

SOLIDITY COMPILER
0.5.17+commit.d19bba13
Include nightly builds
Solidity
compiler default
Auto compile
Enable optimization
Hide warnings
Compile Variable.sol
Solidity_var_Test (Variable.sol)
Publish on Swarm
Publish on Ipfs
Compilation Details
ABI
Bytecode
Variable.sol

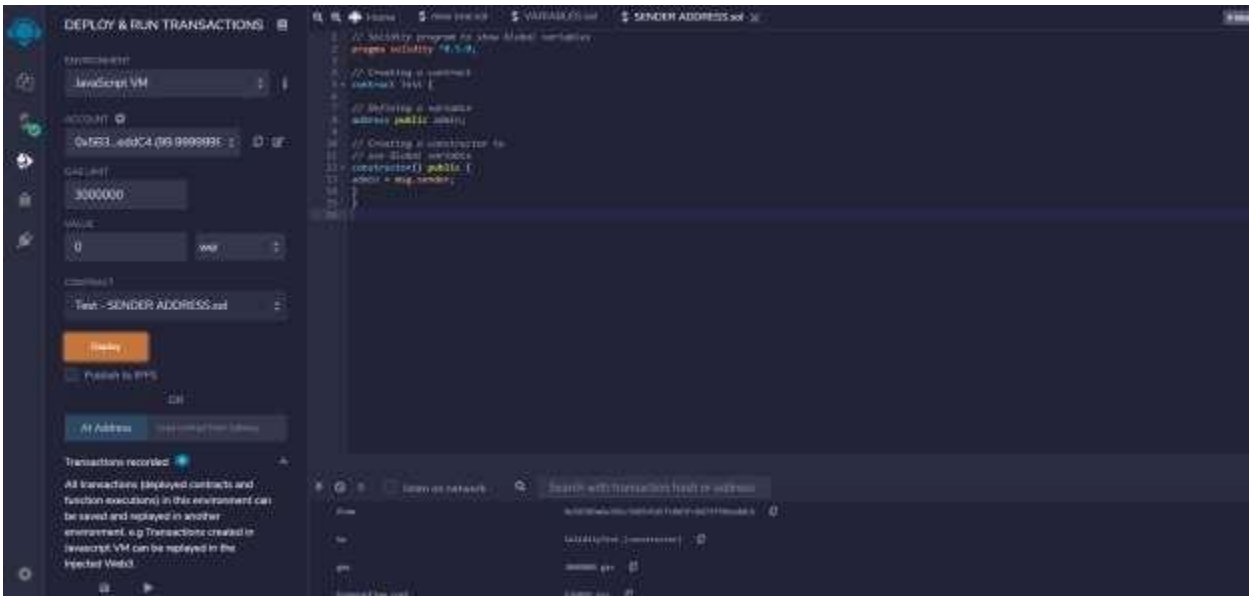

DEPLOY & RUN TRANSACTIONS
JavaScript VM
3000000
0
wei
Solidity_var_Test - Variable.sol
Deploy
Publish to IPFS
OR
At Address
Transactions recorded
Deployed Contracts
state_var
Low level interactions
CALLDATA
Transact
Variable.sol

// **Solidity program to show Global variables**

```
pragma solidity ^0.5.0;

// Creating a contract
contract Test {

// Defining a variable
address public admin;

// Creating a constructor to
// use Global variable
constructor() public {
admin = msg.sender;
}
}
```

Output:



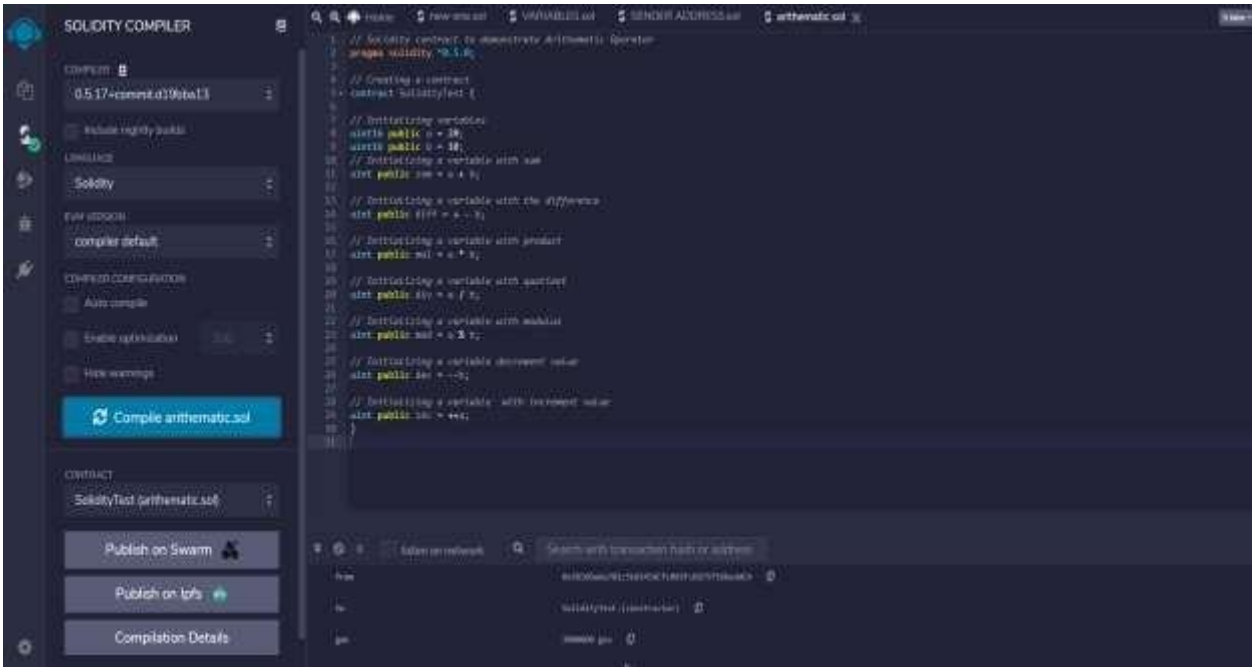

DEPLOY & RUN TRANSACTIONS
JavaScript VM
3000000
Test - SENDER ADDRESS.sol
Deploy
At Address
Transactions recorded
SENDER ADDRESS.sol

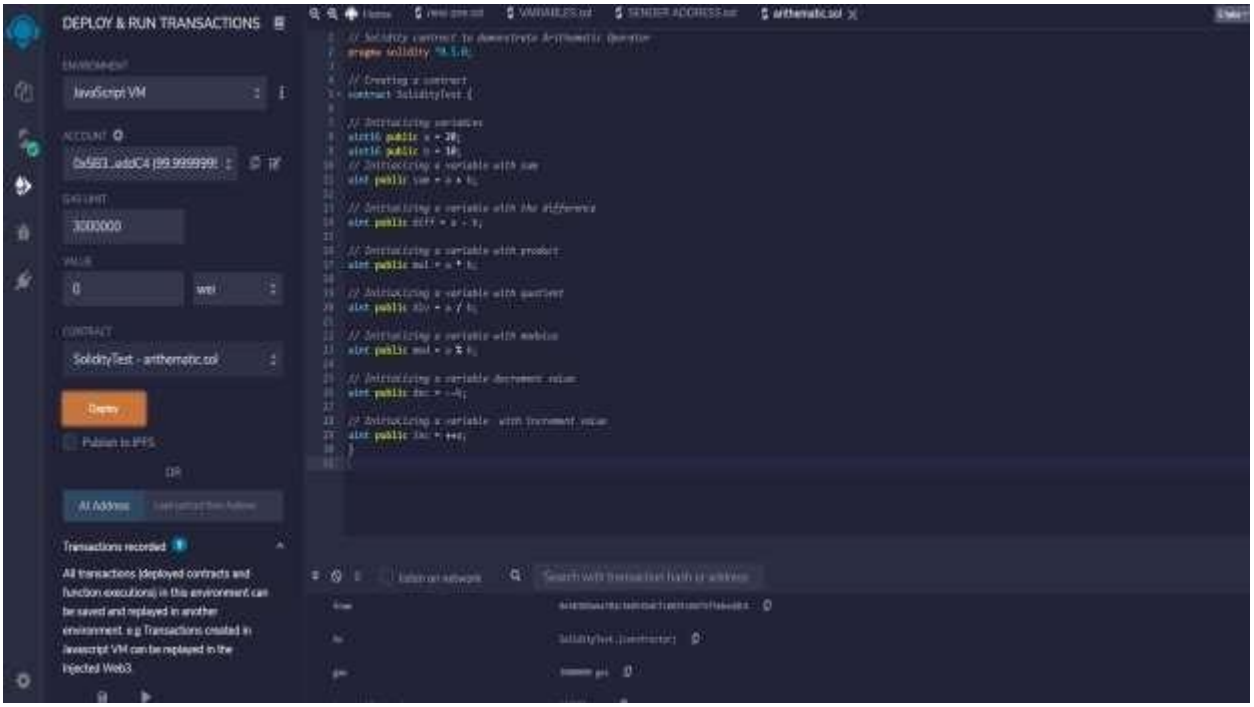
(II). Operators

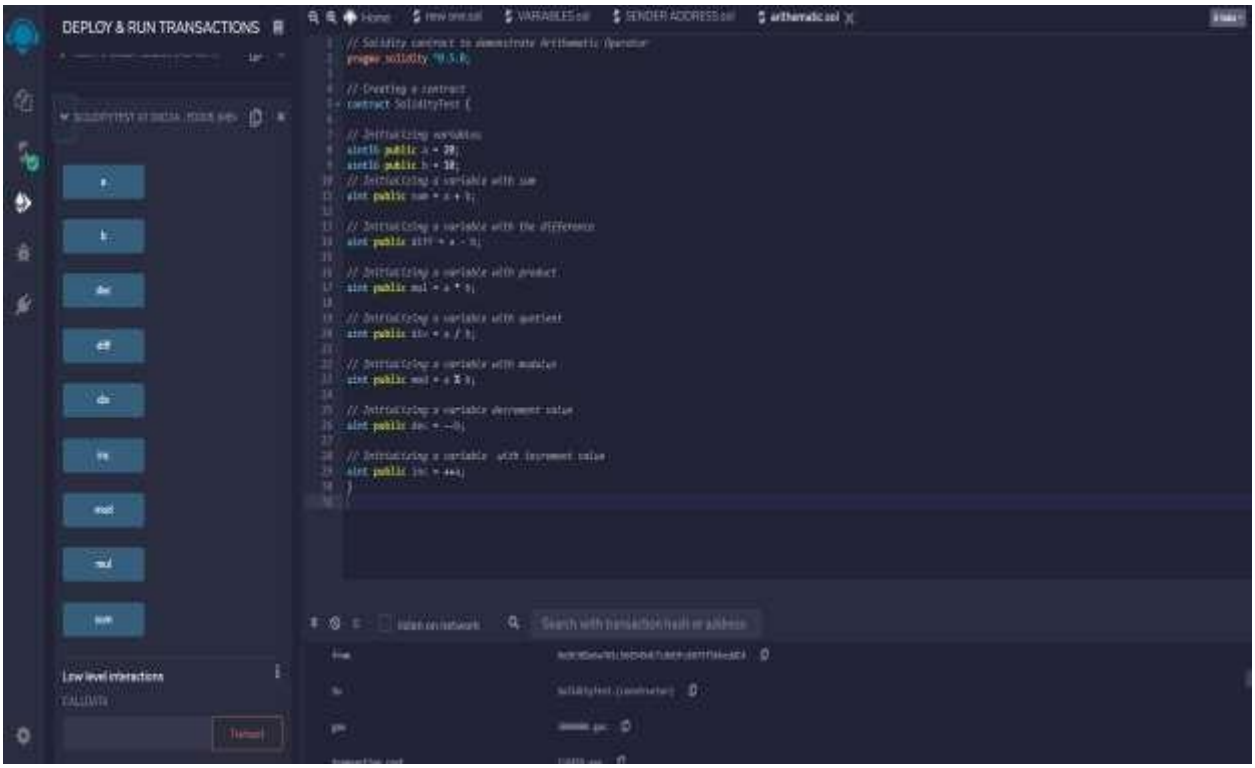
```
// Solidity contract to demonstrate Arithematic Operator
pragma solidity ^0.5.0;

// Creating a contract
contract SolidityTest {

// Initializing variables
uint16 public a = 20;
uint16 public b = 10;
// Initializing a variable with sum
uint public sum = a + b;

// Initializing a variable with the difference
uint public diff = a - b;

// Initializing a variable with product
uint public mul = a * b;

// Initializing a variable with quotient
uint public div = a / b;

// Initializing a variable with modulus
uint public mod = a % b;

// Initializing a variable decrement value
uint public dec = --b;

// Initializing a variable  with increment value
uint public inc = ++a;
```

}

Output:


SOLIDITY COMPILER
Solidity
compiler default
Compile arithematic.sol
SolidityTest (arithematic.sol)
Publish on Swarm
Publish on Ipfs
Compilation Details

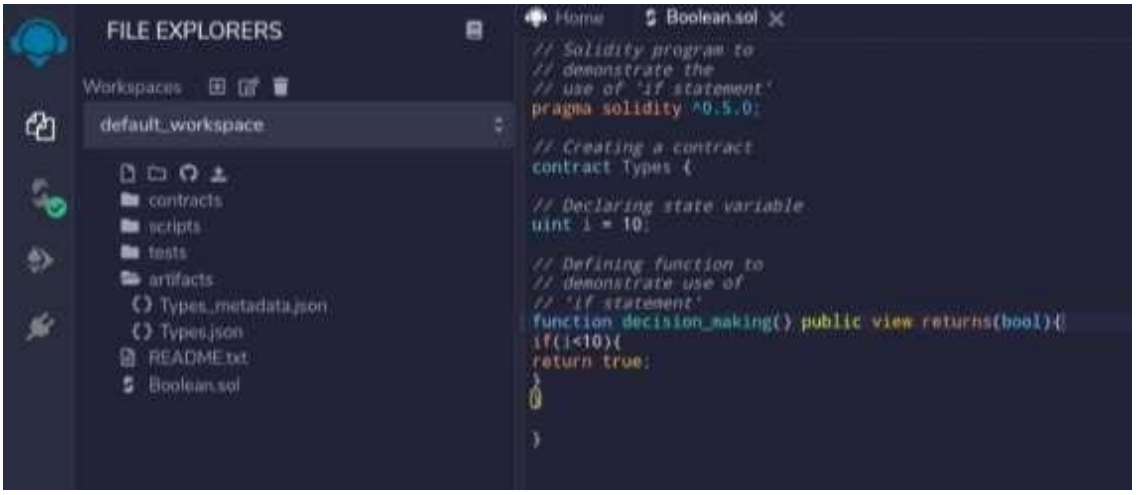

DEPLOY & RUN TRANSACTIONS
JavaScript VM
3000000
SolidityTest - arithematic.sol
Deploy
Transactions recorded

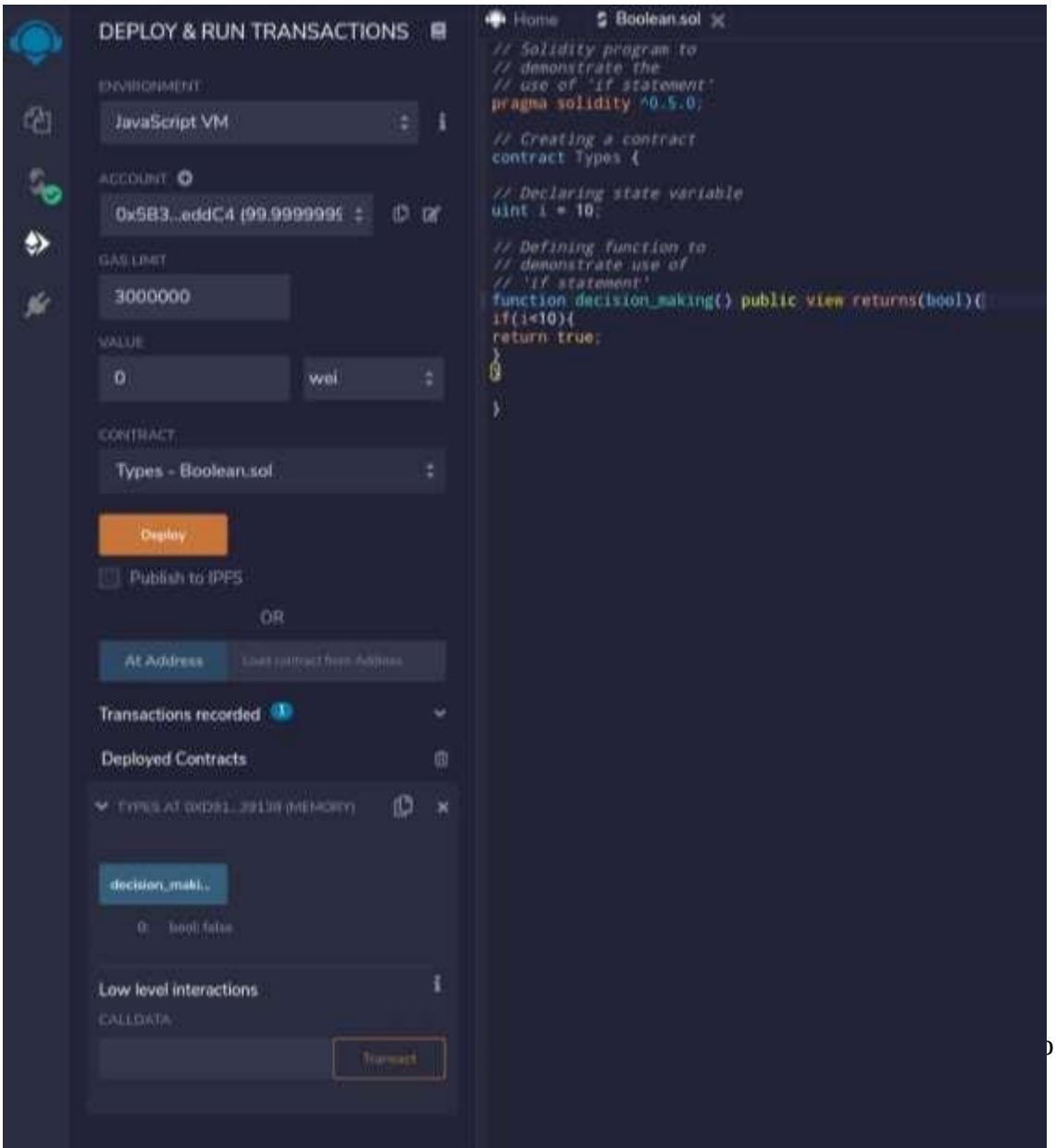

DEPLOY & RUN TRANSACTIONS
Low level interactions
CALLDATA
Transact

(III). Decision Making

```
// Solidity program to demonstrate the use of 'if statement'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {
// Declaring state variable
uint i = 10;

// Defining function to demonstrate use of 'if statement'
function decision_making() public view returns(bool){
if(i<10){
return true;
}
}

}
```

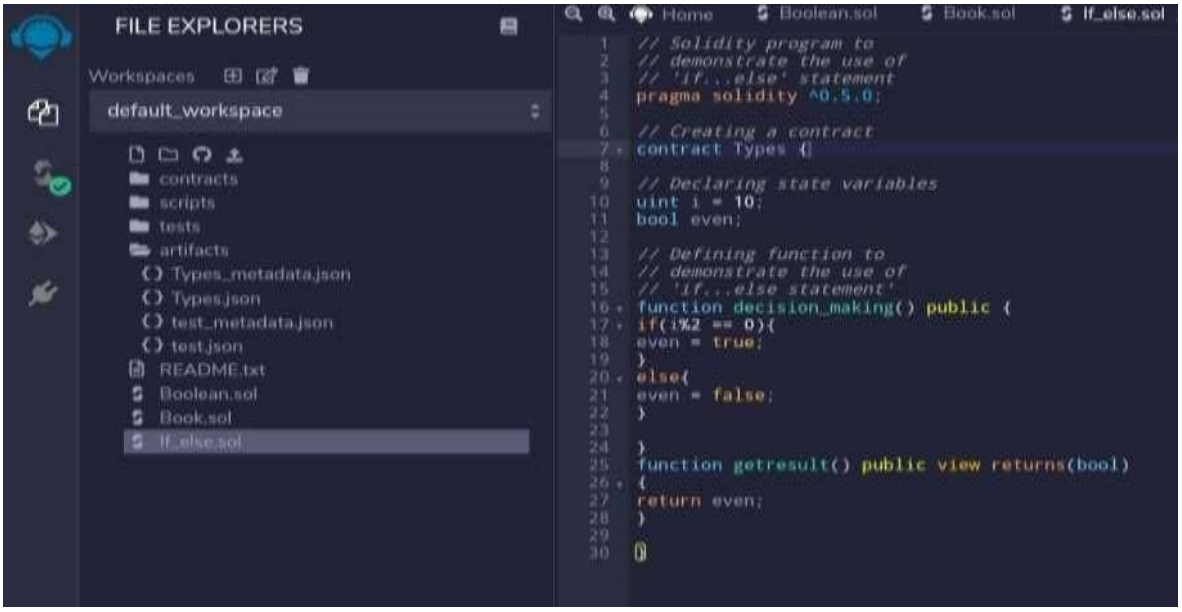
Output:

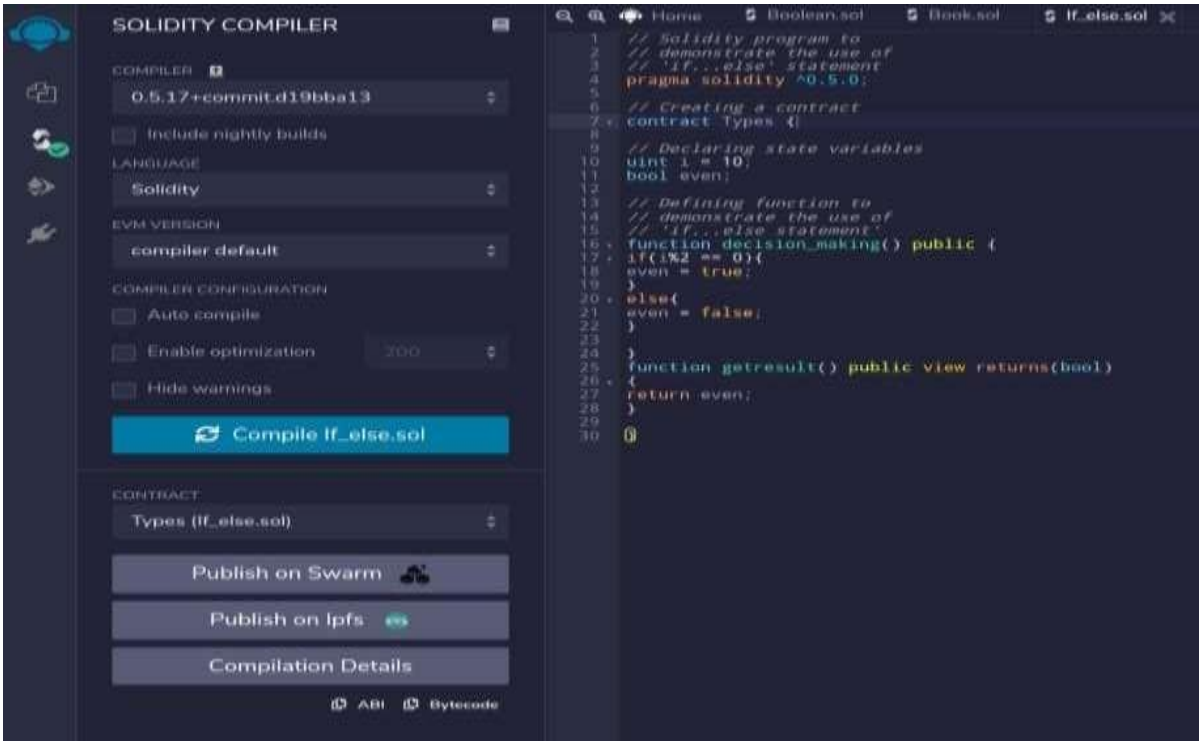

FILE EXPLORERS
Workspaces
default_workspace
contracts
scripts
tests
artifacts
Types_metadata.json
Types.json
README.txt
Boolean.sol
Home
Boolean.sol
// Solidity program to
// demonstrate the
// use of 'if statement'
pragma solidity ^0.5.0;
// Creating a contract
contract Types {
// Declaring state variable
uint i = 10;
// Defining function to
// demonstrate use of
// 'if statement'
function decision_making() public view returns(bool){
if(i<10){
return true;
}
}

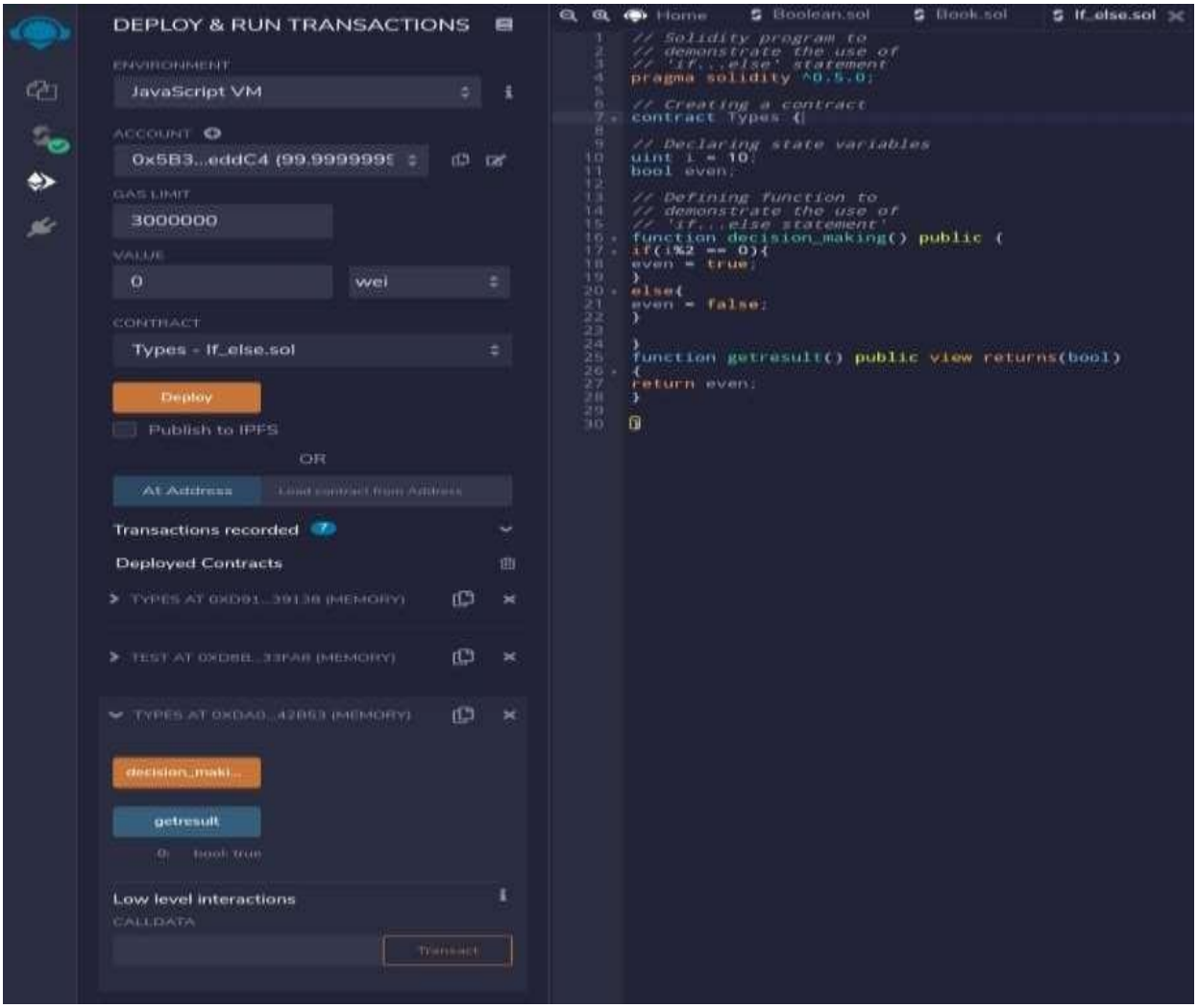

DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
JavaScript VM
ACCOUNT
0x5B3...eddC4 (99.9999995
GAS LIMIT
3000000
VALUE
0
wei
CONTRACT
Types - Boolean.sol
Deploy
Publish to IPFS
OR
At Address
Transactions recorded
Deployed Contracts
decision_maki...
0: bool: false
Low level interactions
CALLDATA
Transact
Home
Boolean.sol
// Solidity program to
// demonstrate the
// use of 'if statement'
pragma solidity ^0.5.0;
// Creating a contract
contract Types {
// Declaring state variable
uint i = 10;
// Defining function to
// demonstrate use of
// 'if statement'
function decision_making() public view returns(bool){
if(i<10){
return true;
}
}

```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

// Declaring state variables
uint i = 10;
bool even;

// Defining function to
// demonstrate the use of
// 'if...else statement'
function decision_making() public {
if(i%2 == 0){
even = true;
}
else{
even = false;
}
}
function getresult() public view returns(bool)
{
return even;
}

}
```

Output:


FILE EXPLORERS
Workspaces
default_workspace
contracts
scripts
tests
artifacts
Types_metadata.json
Types.json
test_metadata.json
test.json
README.txt
Boolean.sol
Book.sol
If_else.sol
Home
Boolean.sol
Book.sol
If_else.sol
// Solidity program to
// demonstrate the use of
// 'if...else' statement
pragma solidity ^0.5.0;
// Creating a contract
contract Types {
// Declaring state variables
uint i = 10;
bool even;
// Defining function to
// demonstrate the use of
// 'if...else statement'
function decision_making() public {
if(i%2 == 0){
even = true;
}
else{
even = false;
}
}
function getresult() public view returns(bool)
{
return even;
}

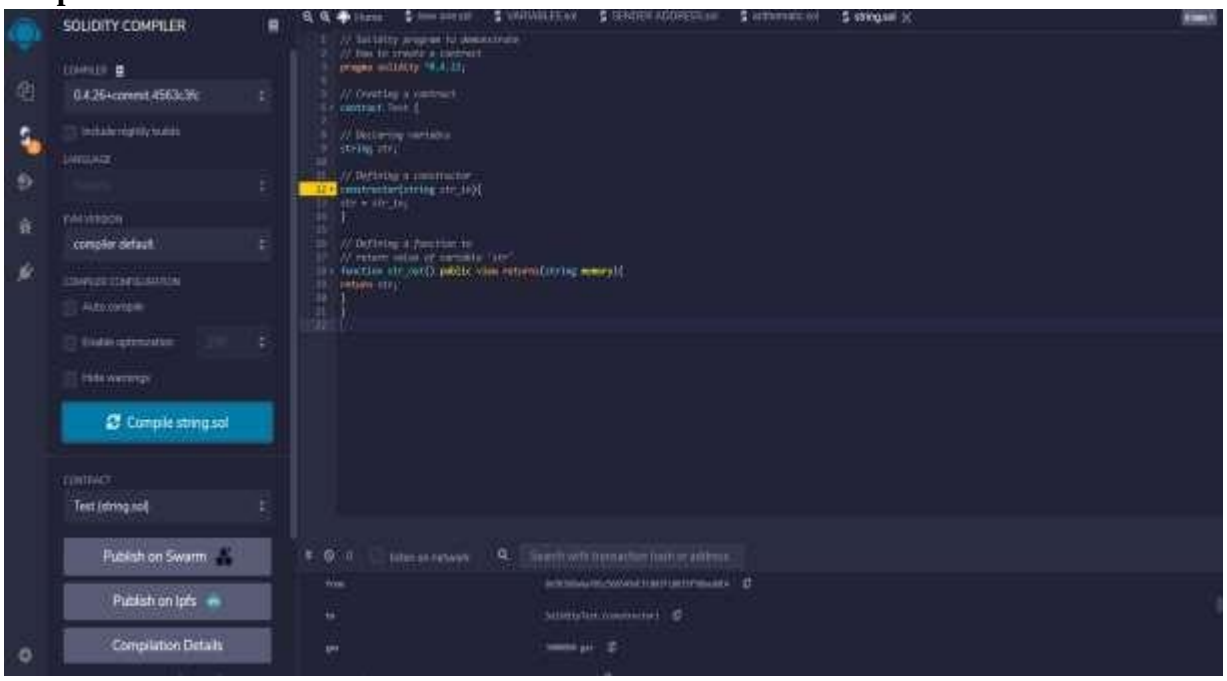

SOLIDITY COMPILER
COMPILER
0.5.17+commit.d19bba13
Include nightly builds
LANGUAGE
Solidity
EVM VERSION
compiler default
COMPILER CONFIGURATION
Auto compile
Enable optimization
200
Hide warnings
Compile If_else.sol
CONTRACT
Types (If_else.sol)
Publish on Swarm
Publish on Ipfs
Compilation Details
ABI
Bytecode
Home
Boolean.sol
Book.sol
If_else.sol
// Solidity program to
// demonstrate the use of
// 'if...else' statement
pragma solidity ^0.5.0;
// Creating a contract
contract Types {
// Declaring state variables
uint i = 10;
bool even;
// Defining function to
// demonstrate the use of
// 'if...else statement'
function decision_making() public {
if(i%2 == 0){
even = true;
}
else{
even = false;
}
}
function getresult() public view returns(bool)
{
return even;
}

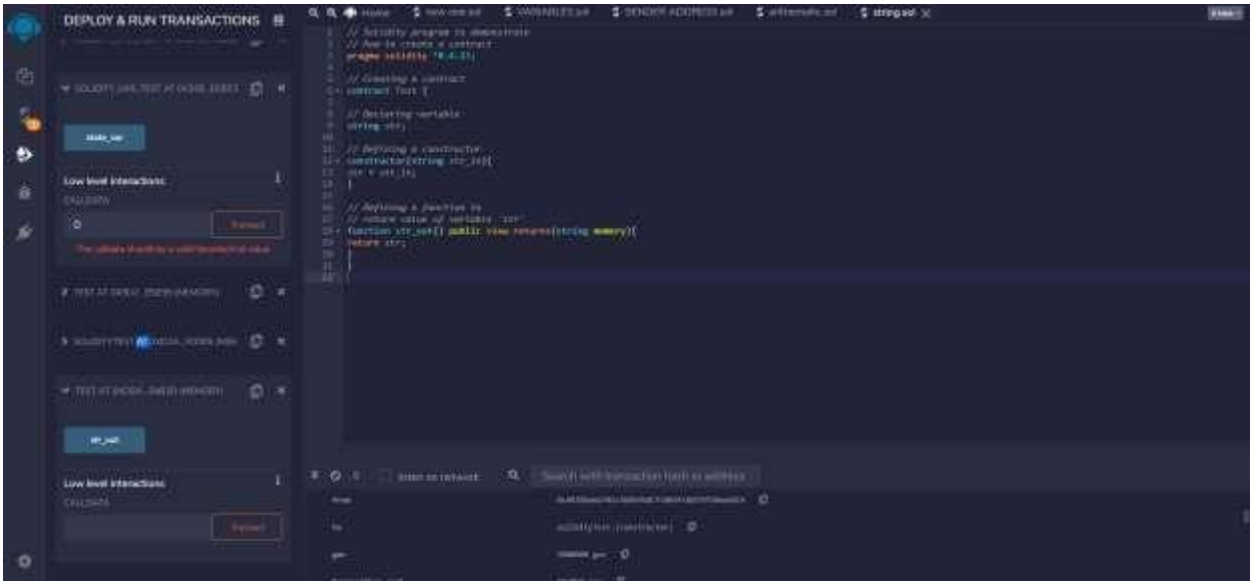

DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
JavaScript VM
ACCOUNT
0x5B3...eddC4 (99.999999
GAS LIMIT
3000000
VALUE
0
wei
CONTRACT
Types - If_else.sol
Deploy
Publish to IPFS
OR
At Address
Transactions recorded
Deployed Contracts
getresult
Low level interactions
CALLDATA
Transact
Home
Boolean.sol
Book.sol
If_else.sol
// Solidity program to
// demonstrate the use of
// 'if...else' statement
pragma solidity ^0.5.0;
// Creating a contract
contract Types {
// Declaring state variables
uint i = 10;
bool even;
// Defining function to
// demonstrate the use of
// 'if...else' statement
function decision_making() public {
if(i%2 == 0){
even = true;
}
else{
even = false;
}
}
function getresult() public view returns(bool)
{
return even;
}

(IV) Strings

```
// Solidity program to demonstrate
// how to create a contract
pragma solidity ^0.4.23;

// Creating a contract
contract Test {

// Declaring variable
string str;

// Defining a constructor
constructor(string str_in){
str = str_in;
}

// Defining a function to
// return value of variable 'str'
function str_out() public view returns(string memory){
return str;
}
}
```

Output


SOLIDITY COMPILER
compiler default
Compile string.sol
Test (string.sol)
Publish on Swarm
Publish on Ipfs
Compilation Details


DEPLOY & RUN TRANSACTIONS
Low level interactions
Low level interactions

Practical 5

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

(I). Arrays

(II). Enums

(III). Structs

(IV). Mappings

(V). Conversations

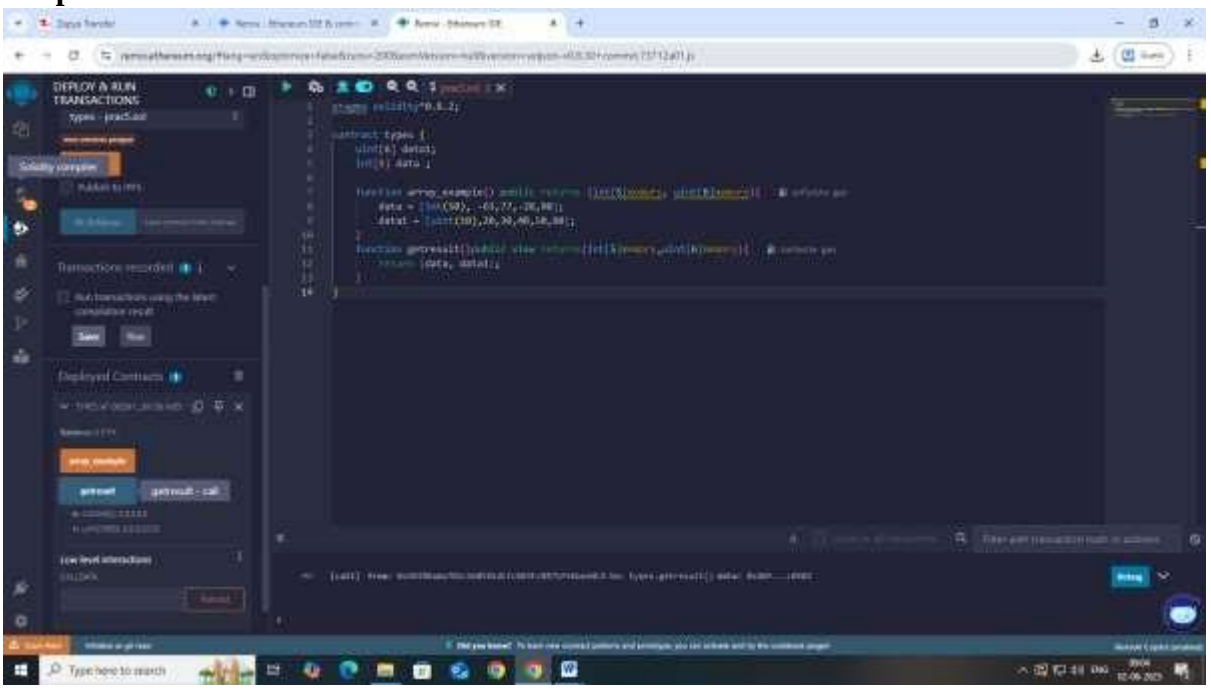
(VI). Ether Units

(VII). Special Variables

(I). Arrays

```
pragma solidity^0.8.2;

contract types {
   uint[6] data1;
   int[5] data ;

   function array_example() public returns (int[5]memory, uint[6]memory){
      data = [int(50), -63,77,-28,90];
      data1 = [uint(10),20,30,40,50,60];
   }
   function getresult()public view returns(int[5]memory,uint[6]memory){
      return (data, data1);
   }
}
```

Output:

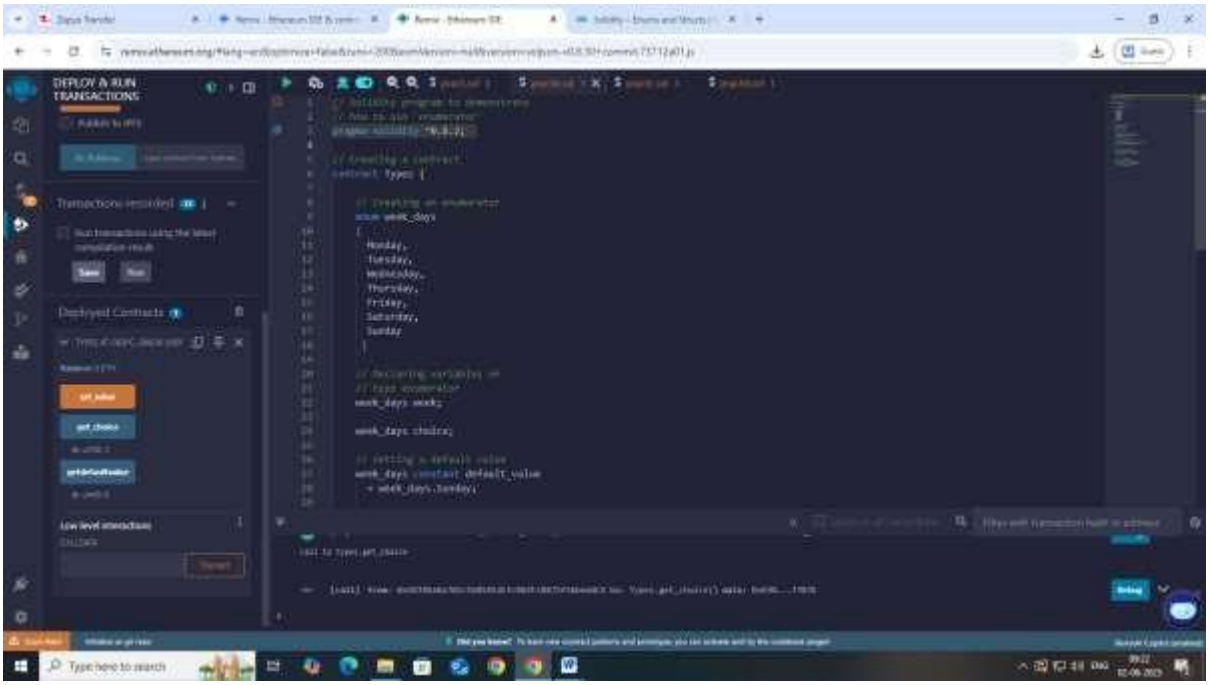

DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions
Type here to search

(II). Enuma

```
// Solidity program to demonstrate
// how to use 'enumerator'
pragma solidity ^0.8.2;

// Creating a contract
contract Types {

    // Creating an enumerator
    enum week_days
    {
      Monday,
      Tuesday,
      Wednesday,
      Thursday,
      Friday,
      Saturday,
      Sunday
     }

    // Declaring variables of
    // type enumerator
    week_days week;

    week_days choice;

    // Setting a default value
    week_days constant default_value
      = week_days.Sunday;

    // Defining a function to
    // set value of choice
    function set_value() public {
      choice = week_days.Thursday;
    }

    // Defining a function to
    // return value of choice
    function get_choice(
    ) public view returns (week_days) {
      return choice;
    }

    // Defining function to
    //  return default value
    function getdefaultvalue(
    ) public pure returns(week_days) {
        return default_value;
    }
```

}

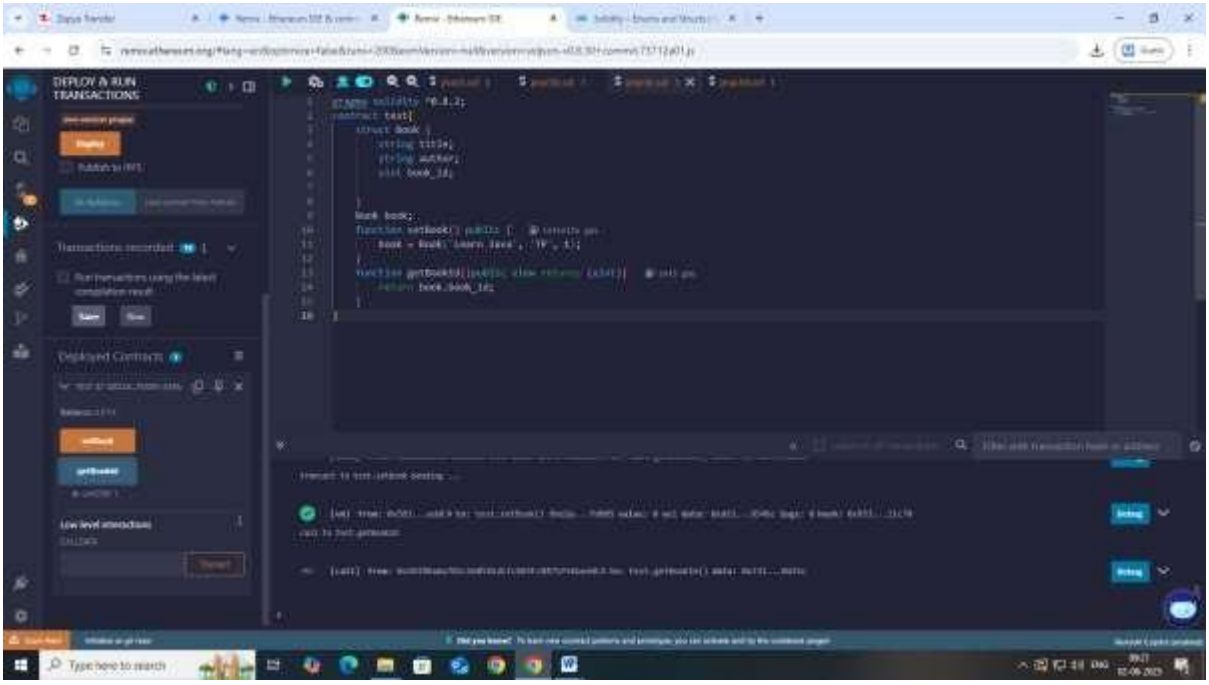
Output:


DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions
Type here to search

(III). Structs

```
pragma solidity ^0.8.2;
contract test{
   struct Book {
      string title;
      string author;
      uint book_id;

   }
   Book book;
   function setBook() public {
      book = Book('Learn Java', 'TP', 1);
   }
   function getBookId()public view returns (uint){
      return book.book_id;
   }
}
```

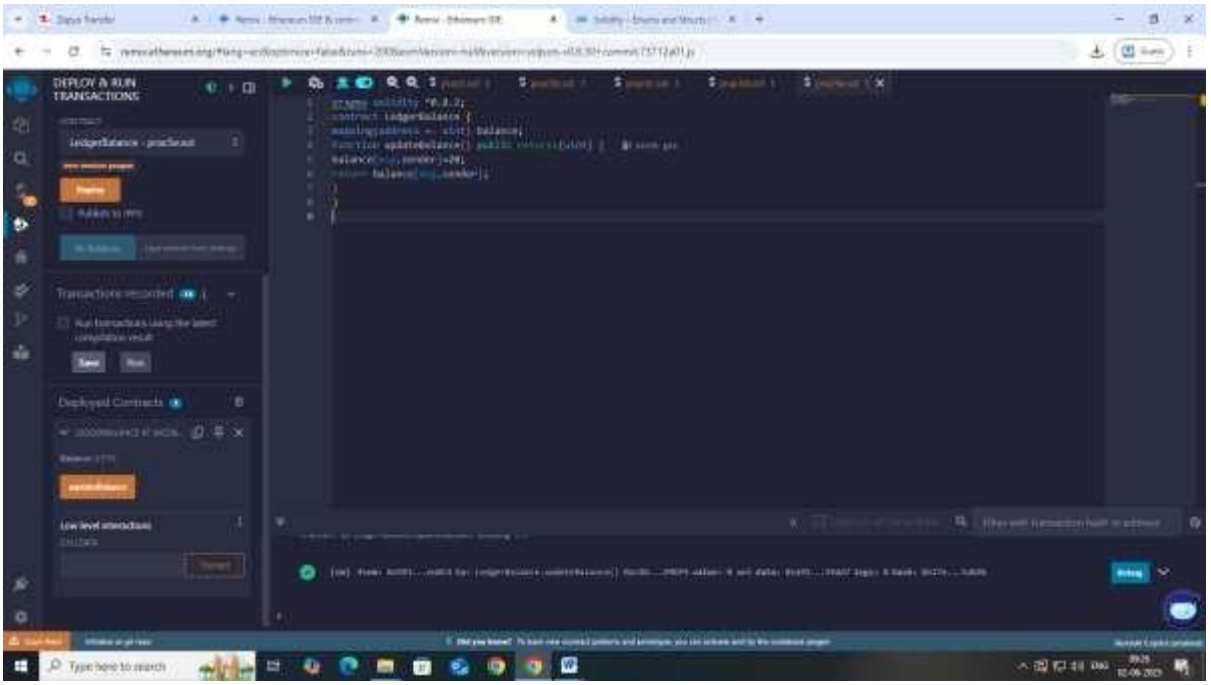
Output:



(IV). Mappings

```
pragma solidity ^0.8.2;
contract LedgerBalance {
mapping(address => uint) balance;
function updateBalance() public returns(uint) {
balance[msg.sender]=20;
return balance[msg.sender];
}
}
```

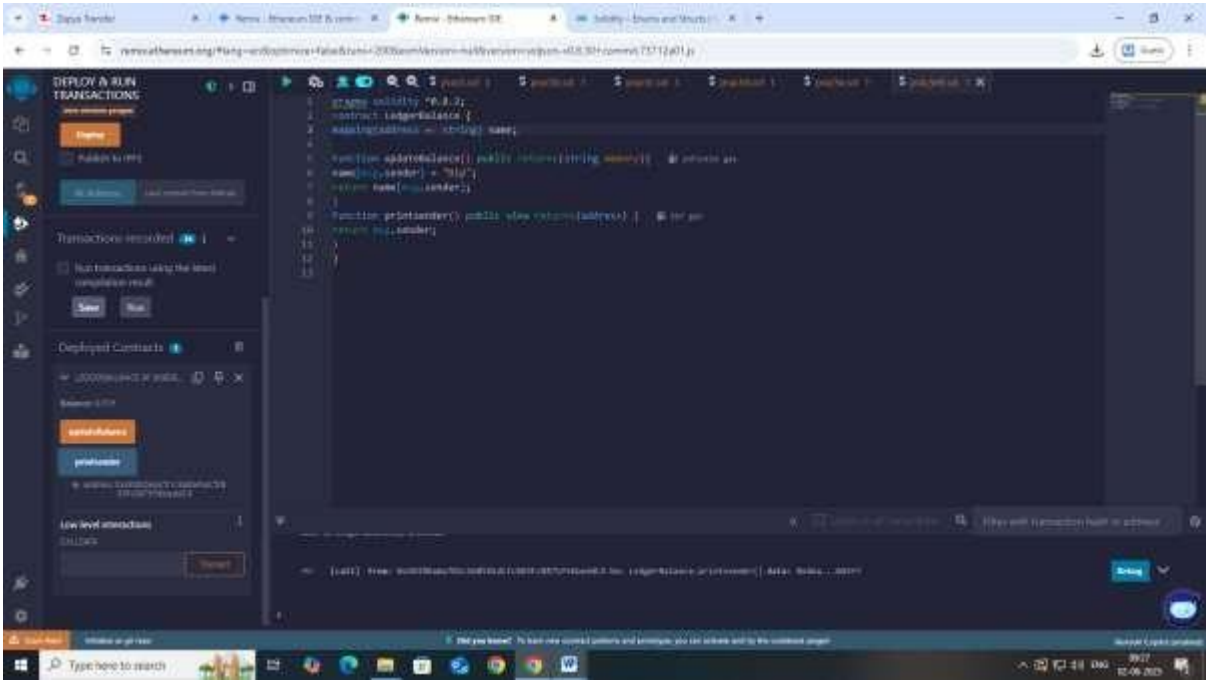
Output:



5e B mapping:

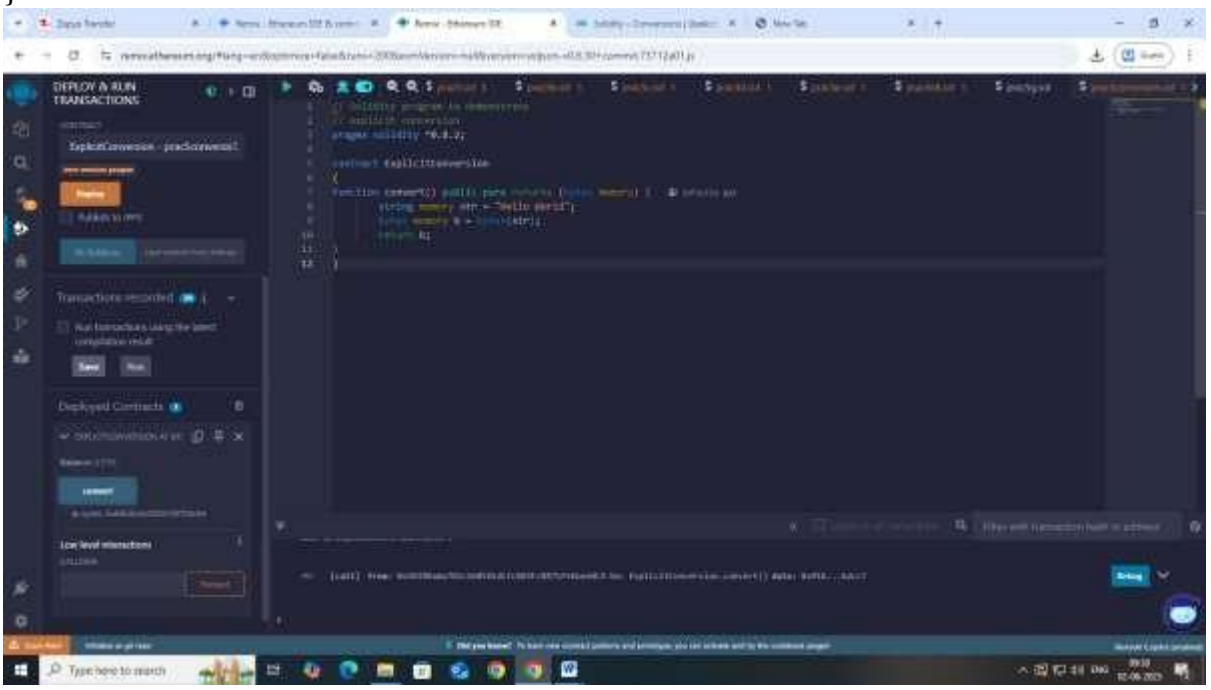
```
pragma solidity ^0.8.2;
contract LedgerBalance {
mapping(address => string) name;

function updateBalance() public returns(string memory){
name[msg.sender] = "Dip";
return name[msg.sender];
}
function printsender() public view returns(address) {
return msg.sender;
}
}
```

Output:


DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions
Type here to search

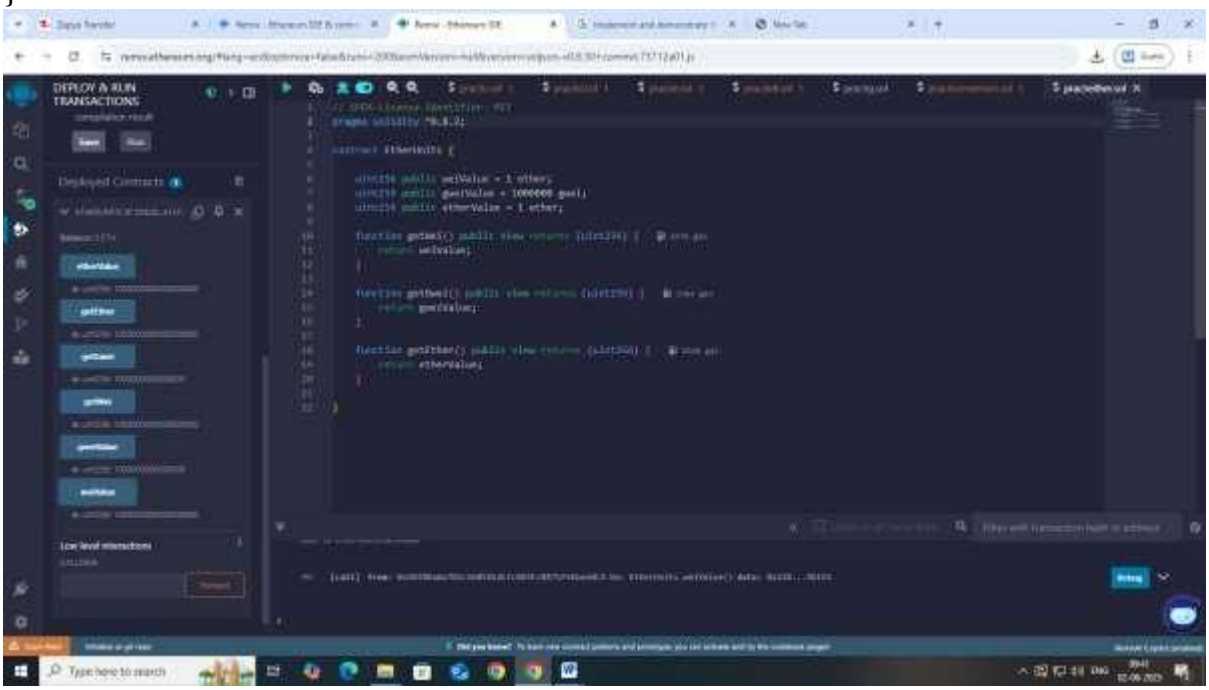
(V). Conversations

```
// Solidity program to demonstrate
// explicit conversion
pragma solidity ^0.8.2;

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



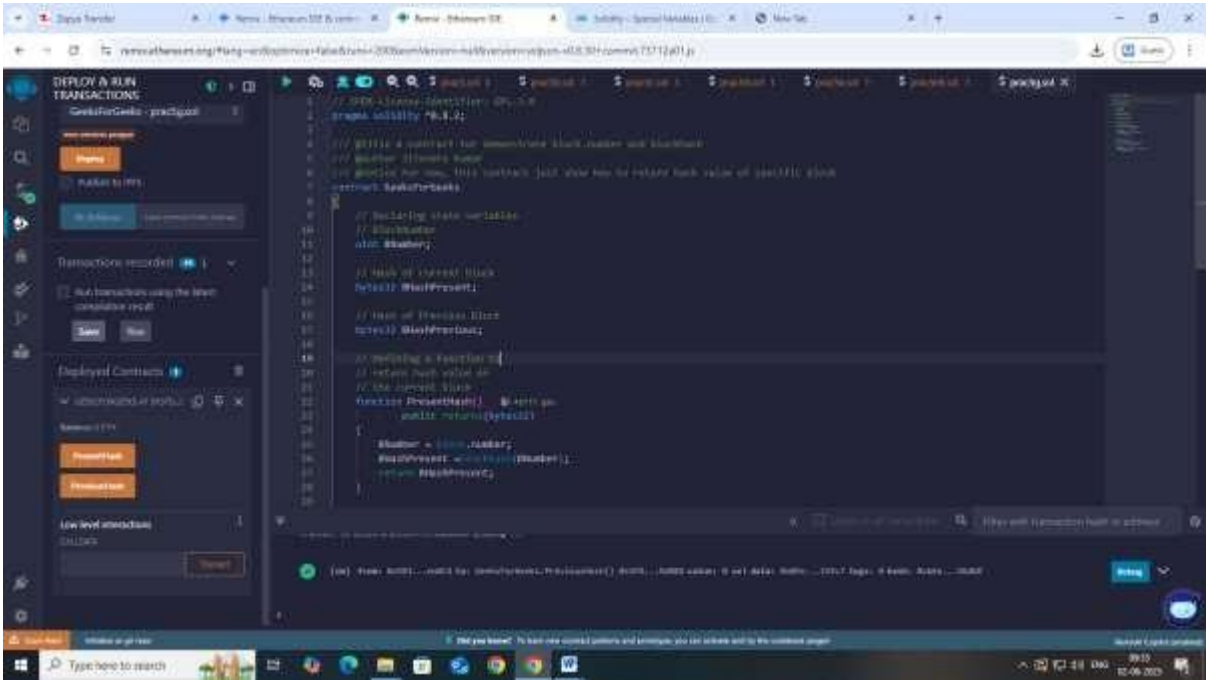
(VI). Ether Units

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

contract EtherUnits {

    uint256 public weiValue = 1 ether;
    uint256 public gweiValue = 1000000 gwei;
    uint256 public etherValue = 1 ether;

    function getWei() public view returns (uint256) {
        return weiValue;
    }

    function getGwei() public view returns (uint256) {
        return gweiValue;
    }

    function getEther() public view returns (uint256) {
        return etherValue;
    }

}
```



(VII). Special Variables

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.2;

/// @title A contract for demonstrate block.number and blockhash
/// @author Jitendra Kumar
/// @notice For now, this contract just show how to return hash value of specific block
contract GeeksForGeeks
{
   // Declaring state variables
   // BlockNumber
   uint BNumber;

   // Hash of current block
   bytes32 BHashPresent;

   // Hash of Previous Block
   bytes32 BHashPrevious;

   // Defining a function to
   // return hash value of
   // the current block
   function PresentHash()
         public returns(bytes32)
   {
      BNumber = block.number;
      BHashPresent =blockhash(BNumber);
      return BHashPresent;
   }

   // Defining a function to
   // return the hash value of
   // the previous block
   function PreviousHash()
         public returns(bytes32)
   {
      BNumber = block.number;
      BHashPrevious =    blockhash(BNumber - 1);
      return BHashPrevious;
   }
}
```


DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions

Practical 6

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

(I). Functions

(II). View Functions

(III). Pure Functions

(IV). Fallback Functions

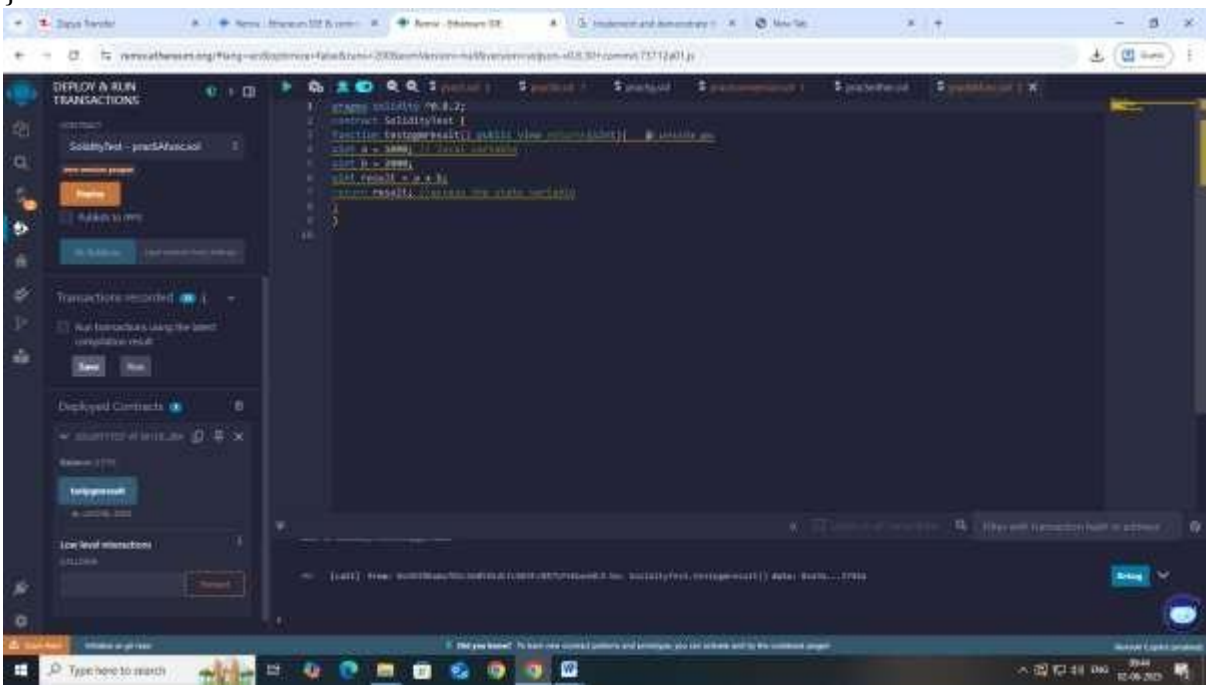
(V). Function Overloading

(VI). Mathematical Functions

(VII). Cryptographic Functions

I). Functions

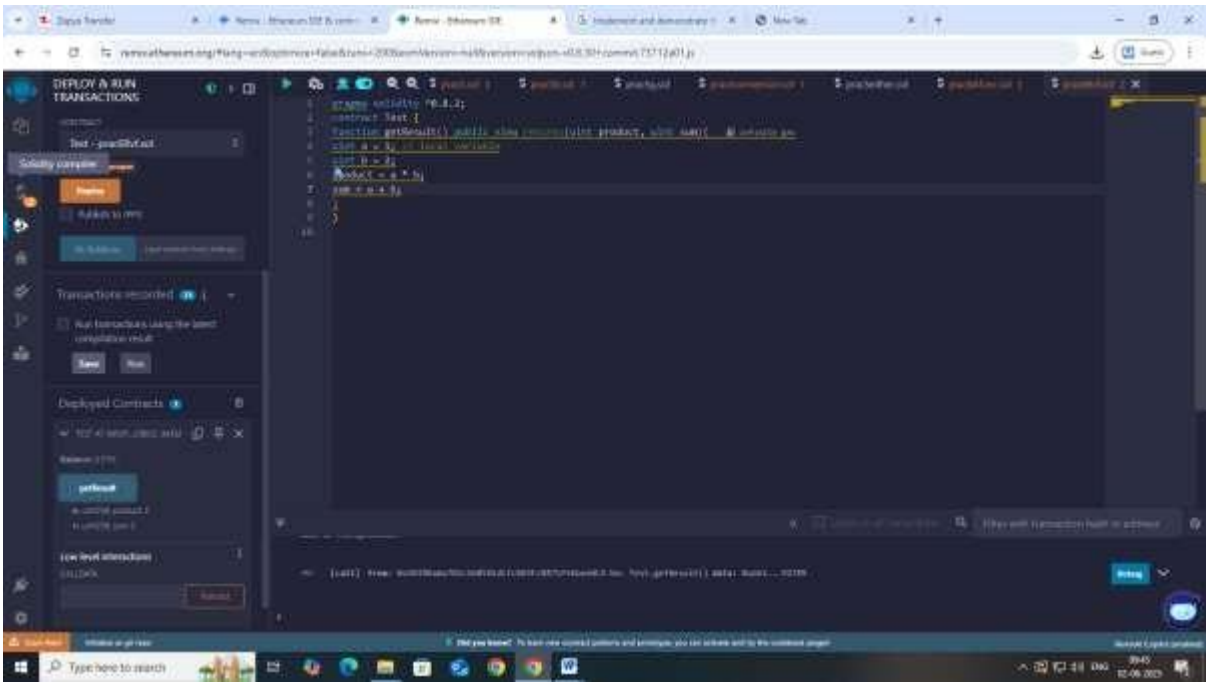
```
pragma solidity ^0.8.2;
contract SolidityTest {
function testpgmresult() public view returns(uint){
uint a = 1000; // local variable
uint b = 2000;
uint result = a + b;
return result; //access the state variable
}
}
```



(II). View Functions

```
pragma solidity ^0.8.2;
contract Test {
function getResult() public view returns(uint product, uint sum){
uint a = 1; // local variable
uint b = 2;
product = a * b;
sum = a + b;
}
}
```

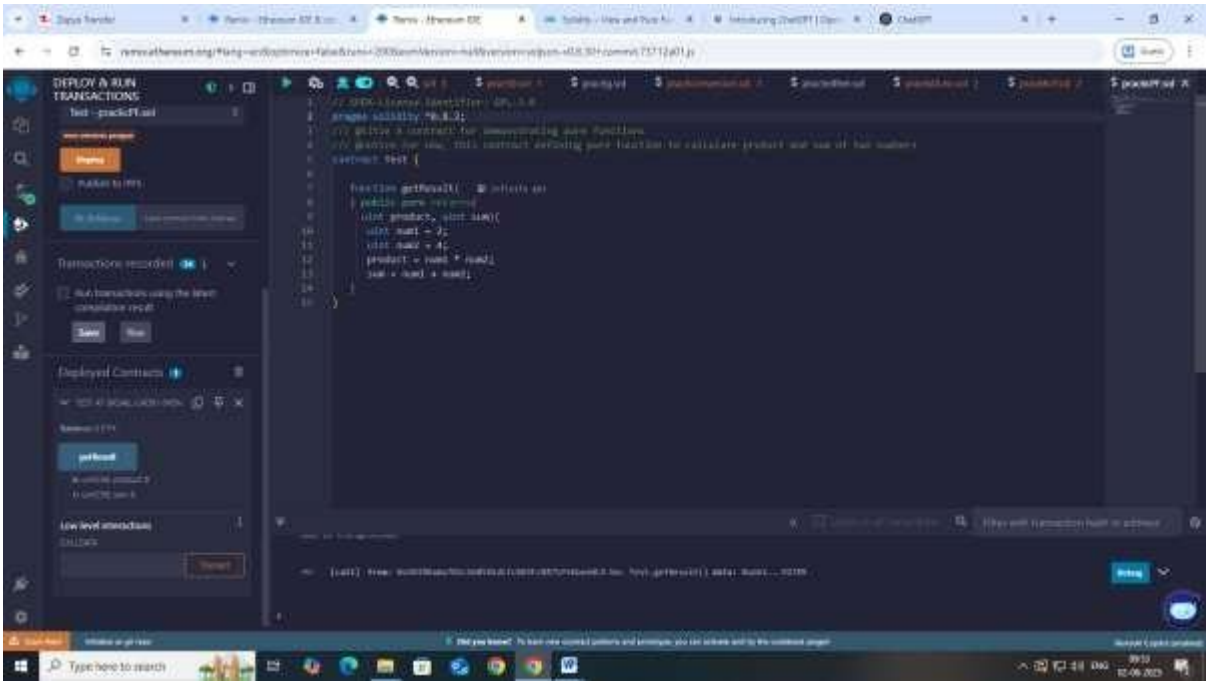
Output:



(III). Pure Functions

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.2;
/// @title A contract for demonstrating pure functions
/// @notice For now, this contract defining pure function to calculate product and sum of two numbers
contract Test {

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

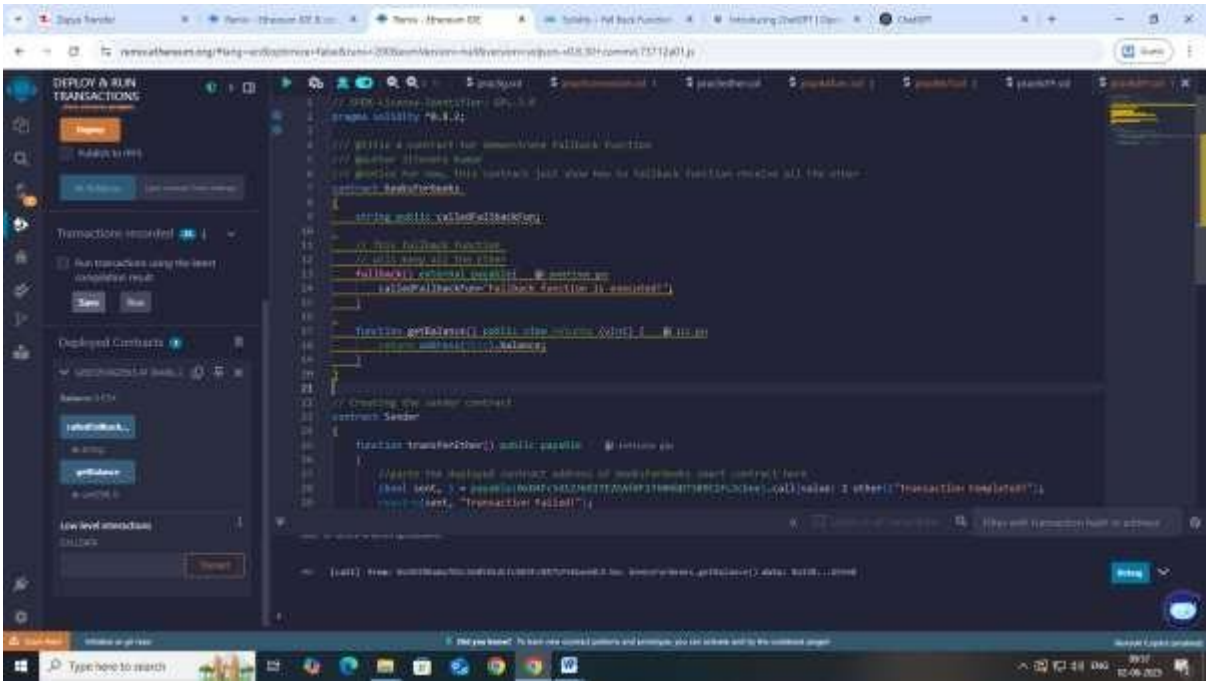
Output:



(IV). Fallback Functions

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.2;

/// @title A contract for demonstrate fallback function
/// @author Jitendra Kumar
/// @notice For now, this contract just show how to fallback function receive all the ether
contract GeeksForGeeks
{
   string public calledFallbackFun;

   // This fallback function
   // will keep all the Ether
   fallback() external payable{
      calledFallbackFun="Fallback function is executed!";
   }

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

// Creating the sender contract
contract Sender
{
   function transferEther() public payable
   {
      //paste the deployed contract address of GeeksForGeeks smart contract here
      (bool sent, ) =
payable(0xD4Fc541236927E2EAf8F27606bD7309C1Fc2cbee).call{value: 2
ether}("Transaction Completed!");
      require(sent, "Transaction Failed!");
   }

   function getBalance() public view returns (uint) {
      return address(this).balance;
   }
}
```

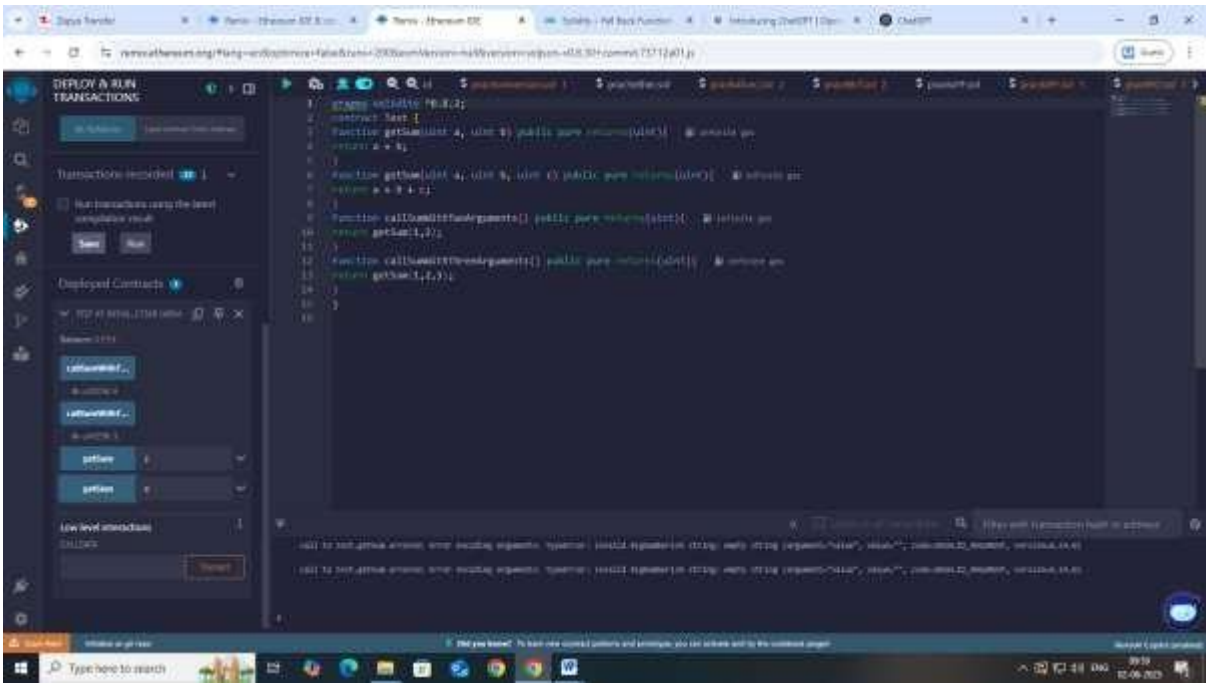
Output:


DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions

(V). Function Overloading

```
pragma solidity ^0.8.2;
contract Test {
function getSum(uint a, uint b) public pure returns(uint){
return a + b;
}
function getSum(uint a, uint b, uint c) public pure returns(uint){
return a + b + c;
}
function callSumWithTwoArguments() public pure returns(uint){
return getSum(1,2);
}
function callSumWithThreeArguments() public pure returns(uint){
return getSum(1,2,3);
}
}
```

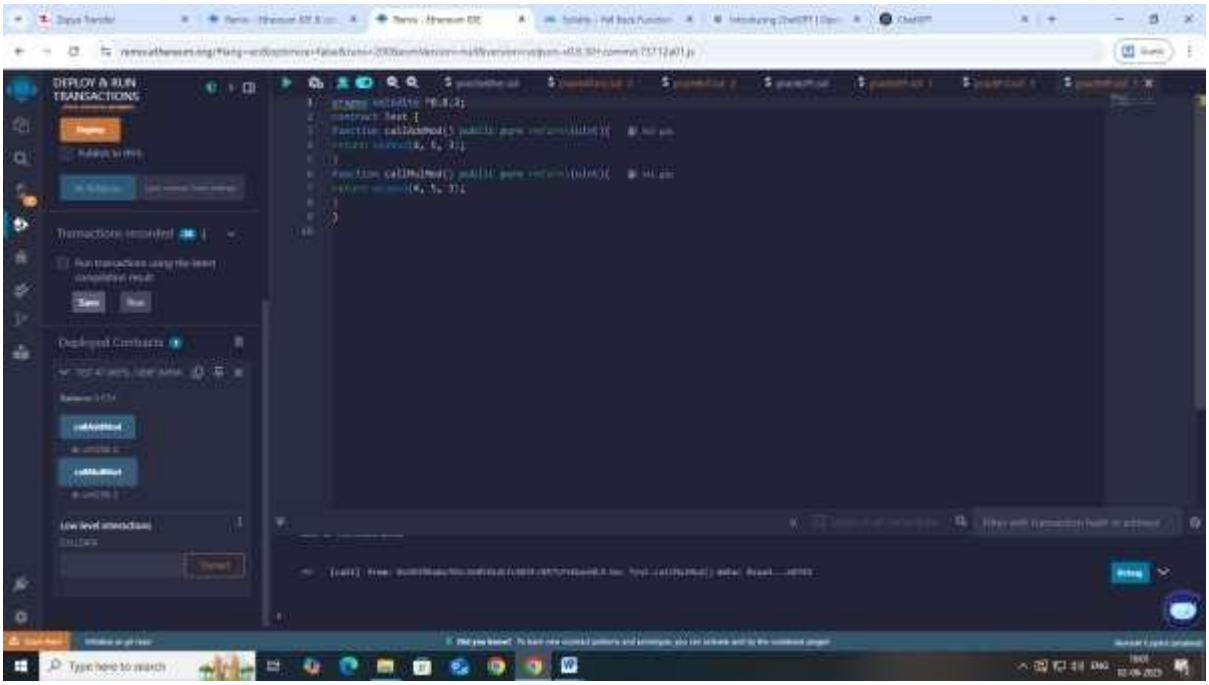
Output:



(VI). Mathematical Functions

```
pragma solidity ^0.8.2;
contract Test {
function callAddMod() public pure returns(uint){
return addmod(4, 5, 3);
}
function callMulMod() public pure returns(uint){
return mulmod(4, 5, 3);
}
}
```

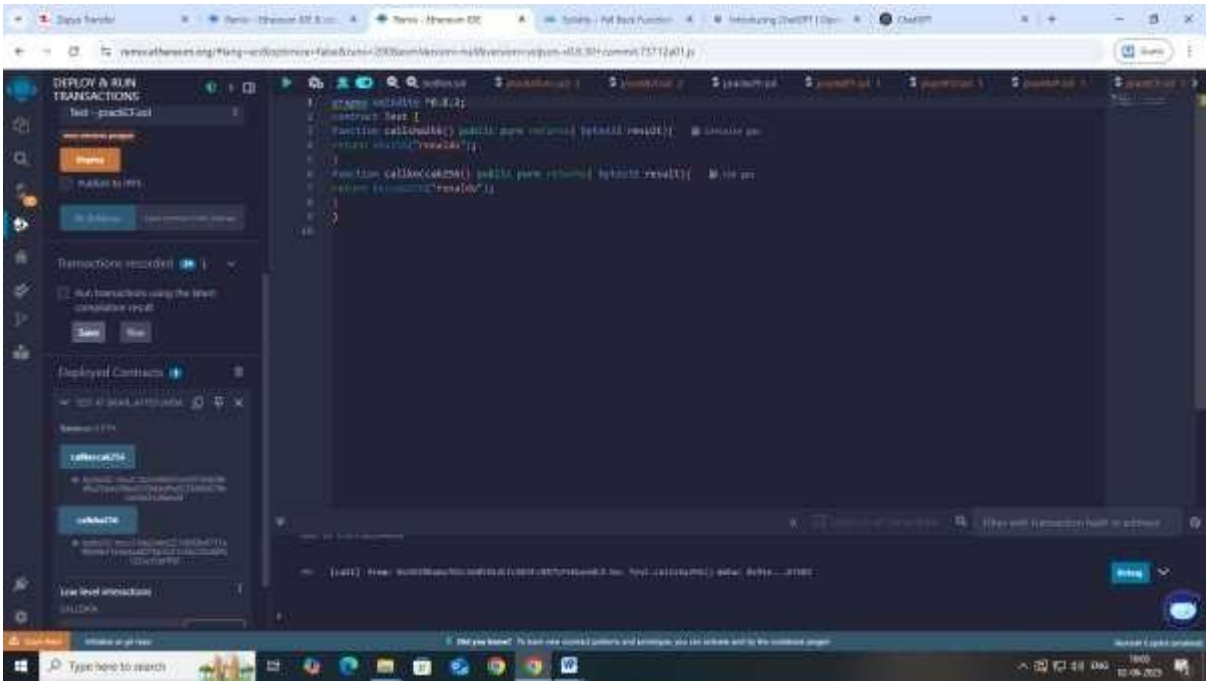
Output:



(VII). Cryptographic Functions

```
pragma solidity ^0.8.2;
contract Test {
function callsha256() public pure returns( bytes32 result){
return sha256("ronaldo");
}
function callkeccak256() public pure returns( bytes32 result){
return keccak256("ronaldo");
}
}
```

Output:



Practical 7

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

(I). Contracts

(II). Inheritance

(III). Constructors

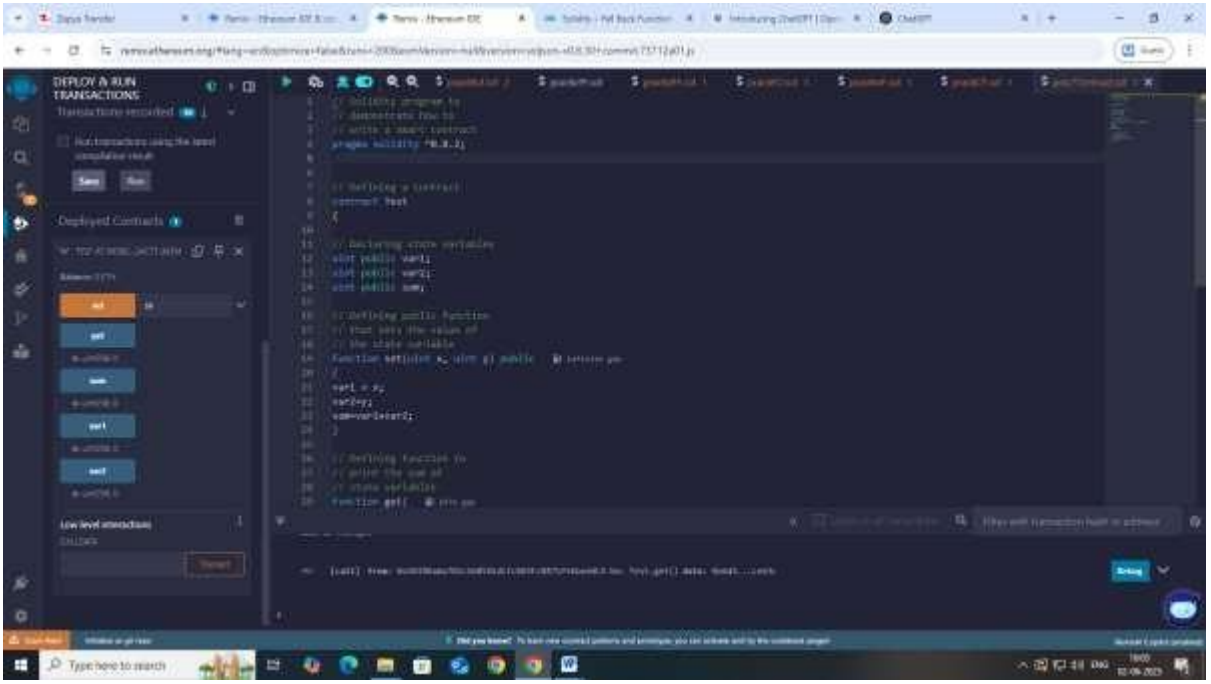
(IV). Abstract Class

(V). Interfaces

(I). Contracts

```
// Solidity program to
// demonstrate how to
// write a smart contract
pragma solidity ^0.8.2;


// Defining a contract
contract Test
{

// Declaring state variables
uint public var1;
uint public var2;
uint public sum;

// Defining public function
// that sets the value of
// the state variable
function set(uint x, uint y) public
{
var1 = x;
var2=y;
sum=var1+var2;
}

// Defining function to
// print the sum of
// state variables
function get(
) public view returns (uint) {
return sum;
}
}
```

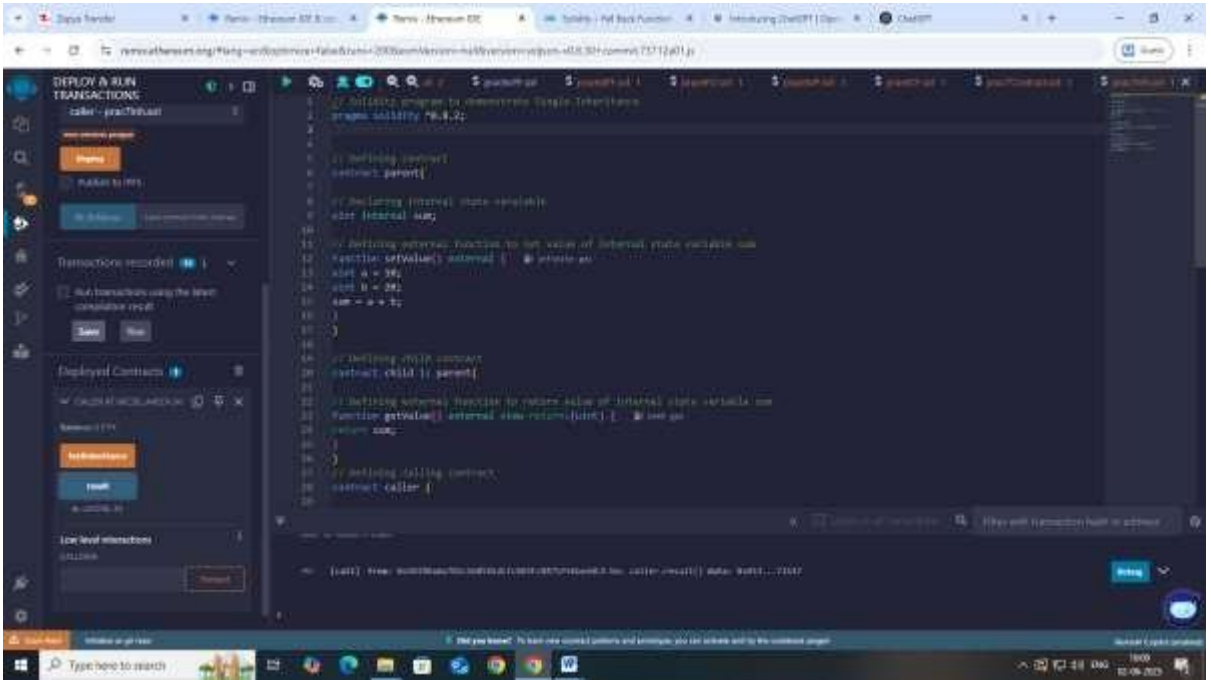
Output:


DEPLOY & RUN TRANSACTIONS
Deployed Contracts
Low level interactions

(II). Inheritance

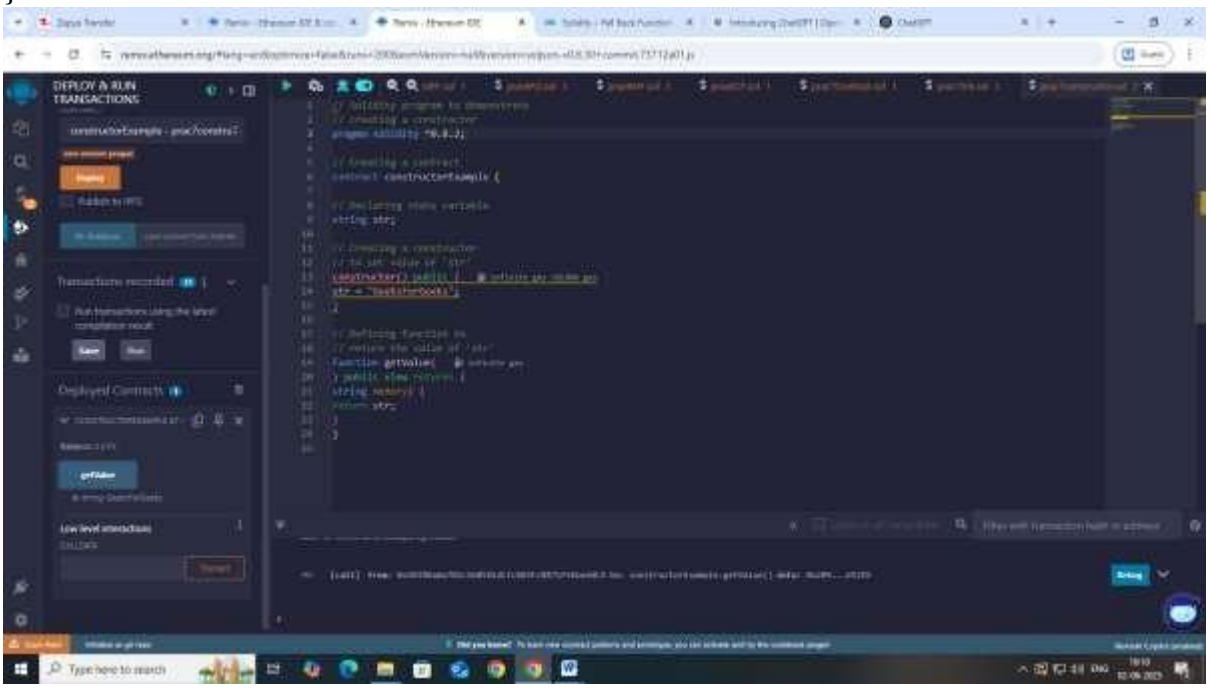
```
// Solidity program to demonstrate Single Inheritance
pragma solidity ^0.8.2;


// Defining contract
contract parent{

// Declaring internal state varaiable
uint internal sum;

// Defining external function to set value of internal state variable sum
function setValue() external {
uint a = 10;
uint b = 20;
sum = a + b;
}
}

// Defining child contract
contract child is parent{

// Defining external function to return value of internal state variable sum
function getValue() external view returns(uint) {
return sum;
}
}
// Defining calling contract
contract caller {

// Creating child contract object
child cc = new child();

// Defining function to call setValue and getValue functions
function testInheritance() public {
cc.setValue();
}
function result() public view returns(uint ){
return cc.getValue();
}
}
```

Output:


DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions

(III). Constructors

```
// Solidity program to demonstrate
// creating a constructor
pragma solidity ^0.8.2;

// Creating a contract
contract constructorExample {

// Declaring state variable
string str;

// Creating a constructor
// to set value of 'str'
constructor() public {
str = "GeeksForGeeks";
}

// Defining function to
// return the value of 'str'
function getValue(
) public view returns (
string memory) {
return str;
}
}
```

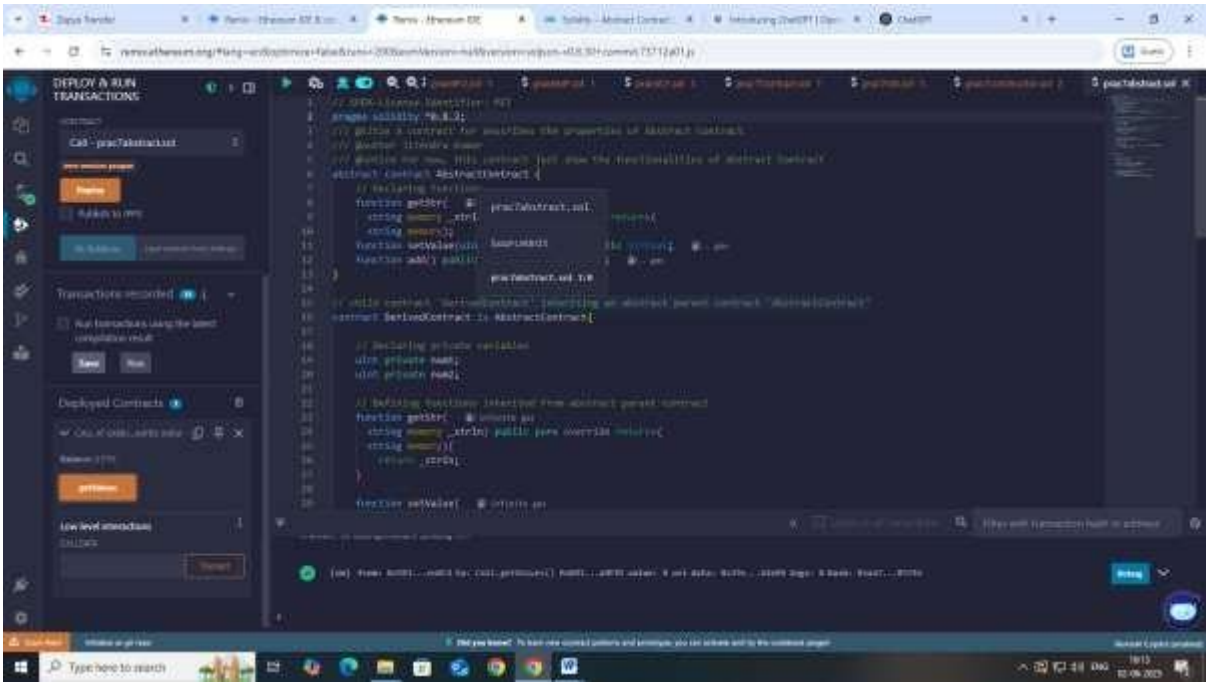


(IV). Abstract Class

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;
/// @title A contract for describes the properties of Abstract Contract
/// @author Jitendra Kumar
/// @notice For now, this contract just show the functionalities of Abstract Contract
abstract contract AbstractContract {
    // Declaring functions
    function getStr(
      string memory _strIn) public view virtual returns(
      string memory);
    function setValue(uint _in1, uint _in2) public virtual;
    function add() public virtual returns(uint);
}

// child contract 'DerivedContract' inheriting an abstract parent contract 'AbstractContract'
contract DerivedContract is AbstractContract{

    // Declaring private variables
    uint private num1;
    uint private num2;

    // Defining functions inherited from abstract parent contract
    function getStr(
      string memory _strIn) public pure override returns(
      string memory){
        return _strIn;
    }

    function setValue(
      uint _in1, uint _in2) public override{
        num1 = _in1;
        num2 = _in2;
    }
    function add() public view override returns(uint){
        return (num2 + num1);
    }
}

// Caller contract
contract Call{

    // Creating an instance of an abstract contract
    AbstractContract abs;

    // Creating an object of child contract
    constructor(){
        abs = new DerivedContract();
```

```
    }

    // Calling functions inherited from abstract contract
    function getValues(
    ) public returns (string memory,uint){
        abs.setValue(10, 16);
        return (abs.getStr("GeeksForGeeks"),abs.add());
    }
}
```

Output:



Practical 8

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

(I) Libraries

(II). Assembly

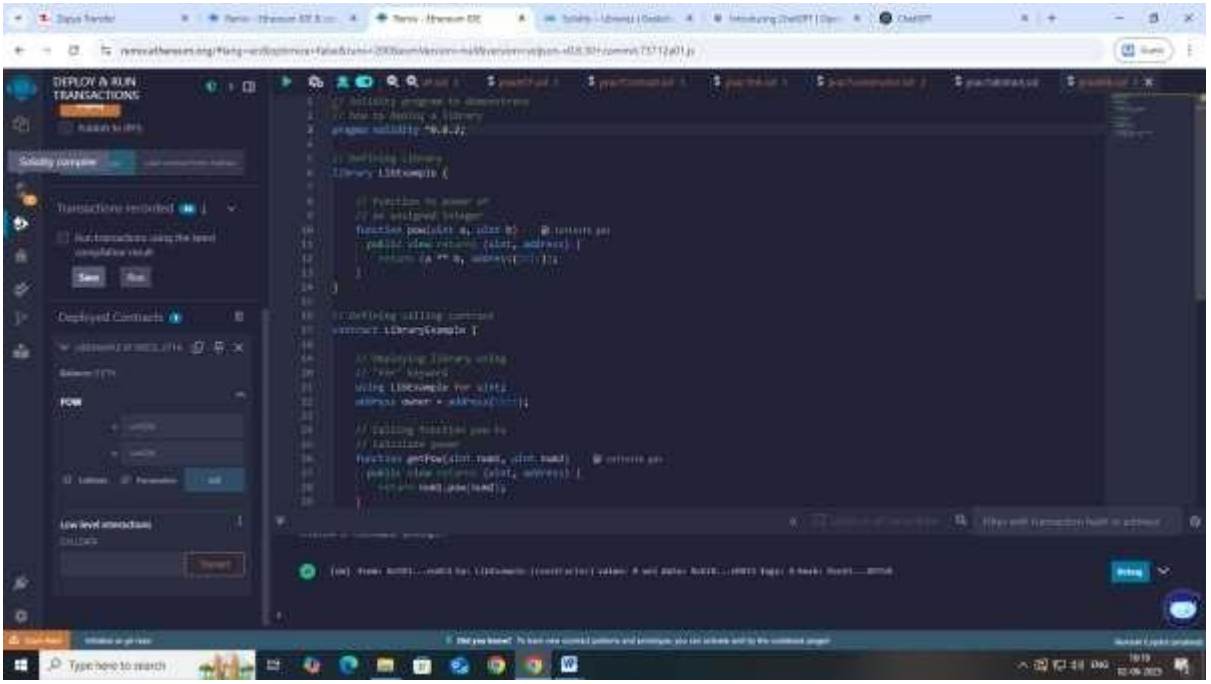
(III) Events

(IV) Error Handling

(I) Libraries

```
// Solidity program to demonstrate
// how to deploy a library
pragma solidity ^0.8.2;

// Defining Library
library LibExample {

    // Function to power of
    // an unsigned integer
    function pow(uint a, uint b)
      public view returns (uint, address) {
        return (a ** b, address(this));
    }
}

// Defining calling contract
contract LibraryExample {

    // Deploying library using
    // "for" keyword
    using LibExample for uint;
    address owner = address(this);

    // Calling function pow to
    // calculate power
    function getPow(uint num1, uint num2)
      public view returns (uint, address) {
      return num1.pow(num2);
    }
}
```

Output:

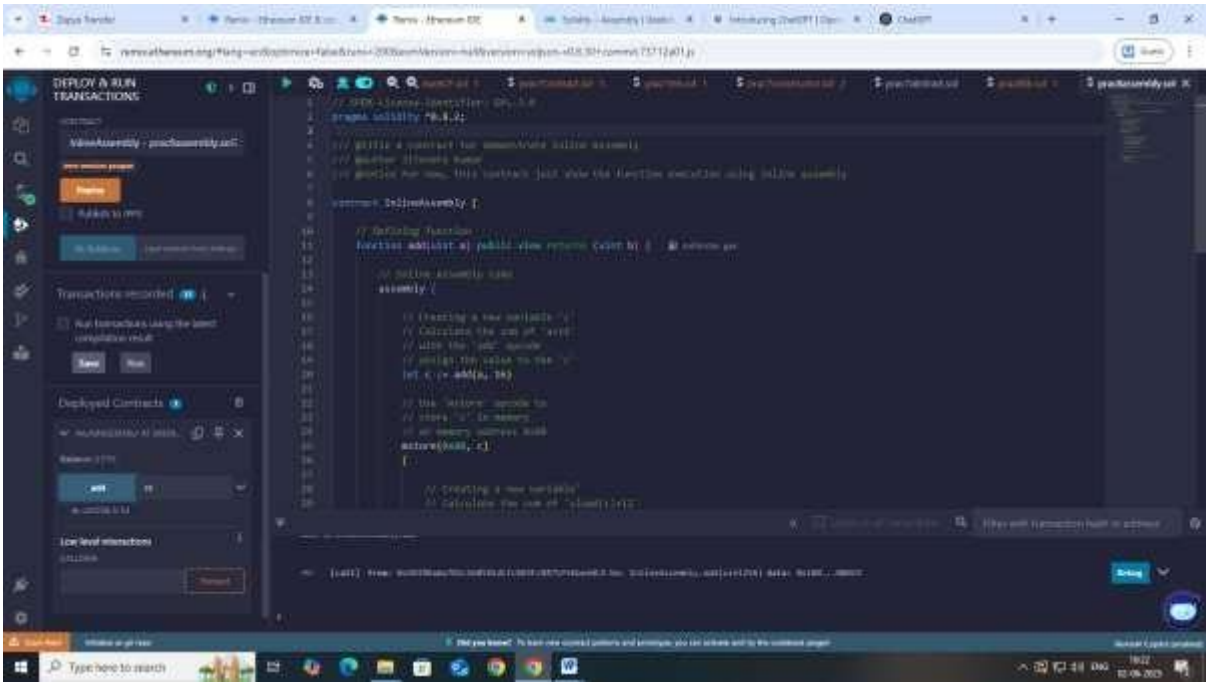

DEPLOY & RUN TRANSACTIONS
Transactions recorded
Deployed Contracts
Low level interactions

(II). Assembly

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.2;

/// @title A contract for demonstrate Inline Assembly
/// @author Jitendra Kumar
/// @notice For now, this contract just show the function execution using inline assembly

contract InlineAssembly {

    // Defining function
    function add(uint a) public view returns (uint b) {

        // Inline assembly code
        assembly {

            // Creating a new variable 'c'
            // Calculate the sum of 'a+16'
            // with the 'add' opcode
            // assign the value to the 'c'
            let c := add(a, 16)

            // Use 'mstore' opcode to
            // store 'c' in memory
            // at memory address 0x80
            mstore(0x80, c)
            {

                // Creating a new variable'
                // Calculate the sum of 'sload(c)+12'
                // means values in variable 'c'
               // with the 'add' opcode
               // assign the value to 'd'
                let d := add(sload(c), 12)

                 // assign the value of 'd' to 'b'
                b := d

            // 'd' is deallocated now
            }

            // Calculate the sum of 'b+c' with the 'add' opcode
            // assign the value to 'b'
            b := add(b, c)
```

```
        // 'c' is deallocated here
        }
    }
}
```

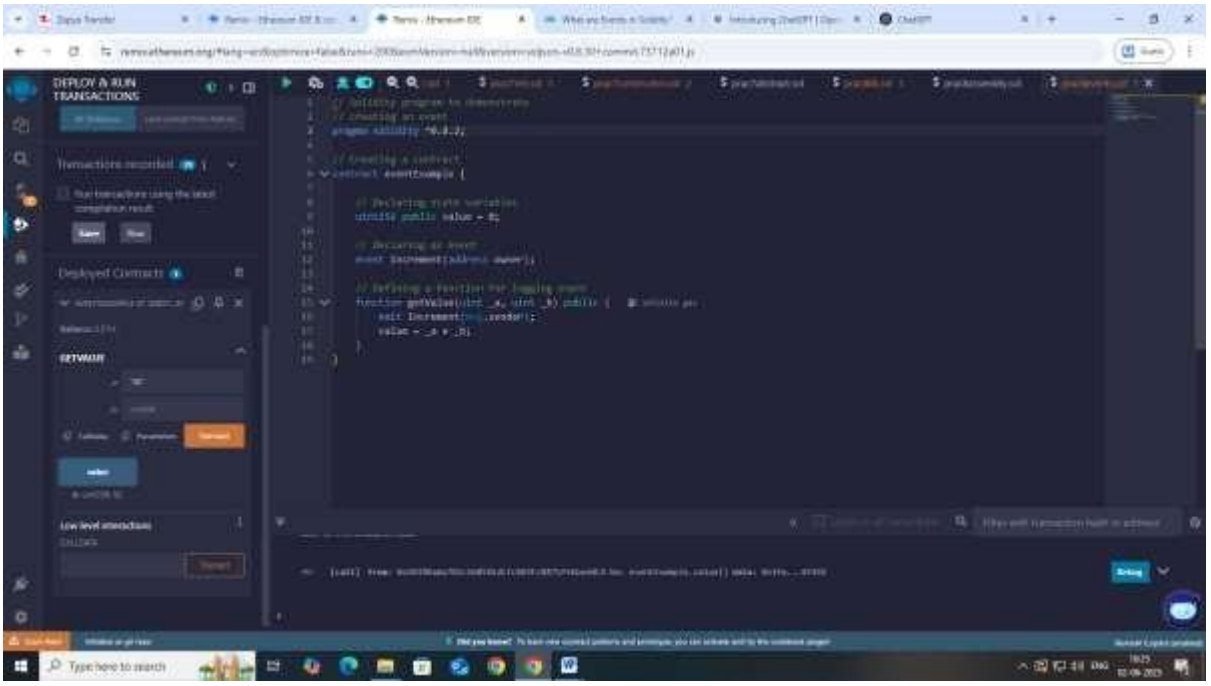
Output:



(III). Events

```
// Solidity program to demonstrate
// creating an event
pragma solidity ^0.8.2;

// Creating a contract
contract eventExample {

    // Declaring state variables
    uint256 public value = 0;

    // Declaring an event
    event Increment(address owner);

    // Defining a function for logging event
    function getValue(uint _a, uint _b) public {
        emit Increment(msg.sender);
        value = _a + _b;
    }
}
```

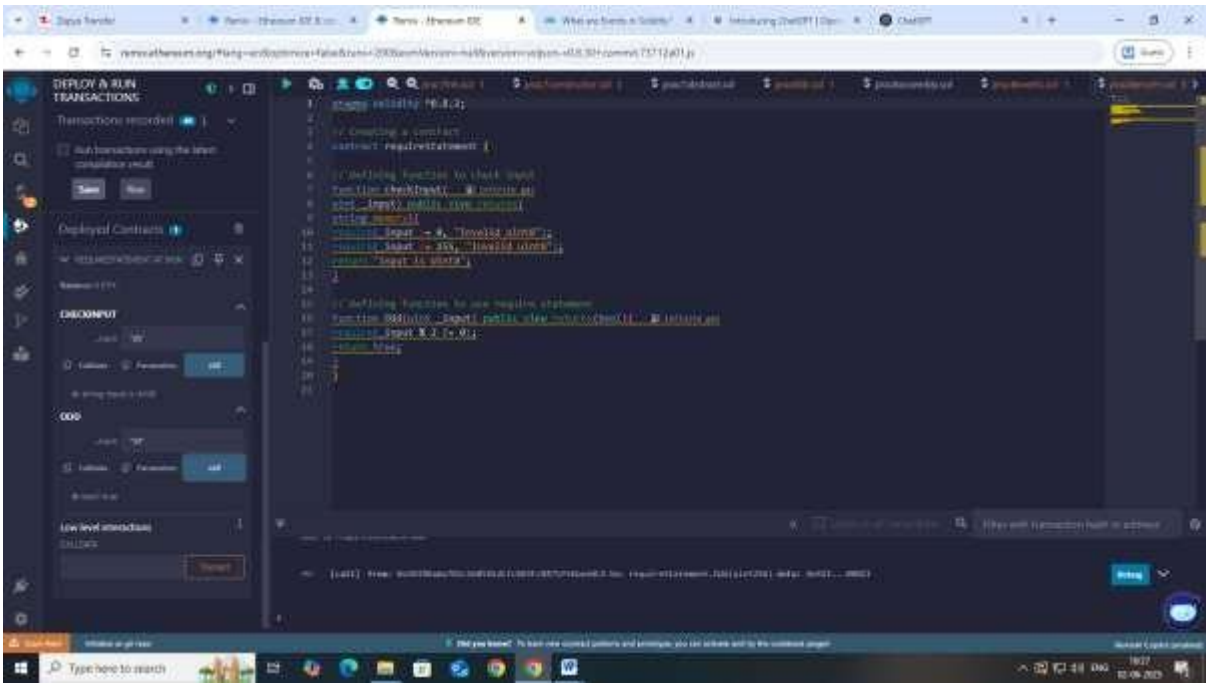
Output:



(IV). Error Handling

```
pragma solidity ^0.8.2;

// Creating a contract
contract requireStatement {

// Defining function to check input
function checkInput(
uint _input) public view returns(
string memory){
require(_input >= 0, "invalid uint8");
require(_input <= 255, "invalid uint8");
return "Input is Uint8";
}

// Defining function to use require statement
function Odd(uint _input) public view returns(bool){
require(_input % 2 != 0);
return true;
}
}
```

Output:



```
// Solidity program to demonstrate assert statement
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

contract AssertExample {
    uint public balance;

    function deposit(uint amount) public {
        balance += amount;
        // After deposit, the balance should always be non-negative.
        assert(balance >= 0);
    }

    function withdraw(uint amount) public {
        // Check if the withdrawal amount is valid.
        require(amount <= balance, "Insufficient balance");
        balance -= amount;
        // After withdrawal, the balance should always be non-negative.
        assert(balance >= 0);
    }

    function transfer(address recipient, uint amount) public {
        // Check if the transfer amount is valid.
        require(amount <= balance, "Insufficient balance");
        balance -= amount;
        // After transfer, the balance should always be non-negative.
        assert(balance >= 0);

        // Verify that recipient address is not zero
```

```
        assert(recipient != address(0));

        // Send the amount to the recipient (This is a simplified example).
        // In a real-world scenario, you'd likely use a payable transfer.
    }

    function add(uint a, uint b) public pure returns (uint) {
        uint result = a + b;
        // Check for overflow
        assert(result >= a);
        return result;
    }
}
```

Output: