PRACTICAL 1

AIM - Write the following programs for Blockchain in Python:

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

Code :

```
#Practical1A
import binascii
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto import Random

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("asci
i")

# Create an instance of the Client class
UDIT = Client()

# Print the public key (identity) of the client
print("\nPublic Key:", UDIT.identity)
```

Output :

```
C:\Users\murarilal\Desktop\Blockchain\ethermine>python Practical1A.py

Public Key: 30819f300d06092a864886f70d010101050003818d0030818902818100e59
c20238a46697904e3b999b6e54351ec6557d53cbf76af9bad70ff6c569571cf037c3c9de3
7d7f68f216b0af52d820a735a47969b3aa44a84b205707783e78b4623795d030428065732
a4d61d41ee4b405a2011b16bad18a819225156a255b0baa254ffbfb0282f16bbddc5ac8d7
ac9ce41b3b734e871304eb3127d0b2f6850203010001
```

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

Code :

```
#Practical1B
import binascii
import collections
import datetime
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto import Random

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("asci
i")


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):
        identity = "Genesis" if self.sender == "Genesis" else
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")

UDIT = Client()
UGC = Client()

t = Transaction(UDIT, UGC.identity, 5.0)

print("\nTransaction Recipient:\n", t.recipient)  #
print("\nTransaction Sender:\n", t.sender) print("\nTransaction
Value:\n", t.value)
signature = t.sign_transaction()
print("\nSignature:\n", signature)
```

Output :

```
C:\Users\murarilal\Desktop\Blockchain\ethermine>python Practical1B.py

Transaction Recipient:
 30819f300d06092a864886f70d010101050003818d0030818902818100bb7ddbc0f1c7a1
ddd848576a531c8395342ae64b440f4d43212ec3c7687cd0f713d7fb2a38e5210c6af5a87
744865f97795b8489d2598ab37a7dea57d1ebb6441a7dcec6464b3b77b9e4e245e21d9e82
aa121abbfdeb74c5c9b4b076872379c20b7cde59826dd50587b124d58736496f097a54e6a
eda9f3d35173300e0e36d910203010001

Signature:
 800d86e1119dd6647db208e61545e36304dd7e18533e1a8465e5ed7eab20b335571eb52b
74a8b1bd18a586ea407c9d94a7c55a3ced0b5932dc91666e225712c9b67a1a6693f01bb6c
90488469f38c3cf43b2159597a6f7e90895da32e65225c73bbee1c8685cd15c69cf9d6aa3
ec6b2958ba9949fd069e526d526de7bd841171
```

[C] Create multiple transactions and display them.

Code :

```
# Practical1 C
import binascii
import collections
import datetime
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto import Random


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("asci
i")


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):
        identity = "Genesis" if self.sender == "Genesis" else
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")

    def display_transaction(transaction):
        # for transaction in transactions:
        dict = transaction.to_dict()
        print("sender: " + dict['sender'])
        print('-----')
        print("recipient: " + dict['recipient'])
        print('-----')
        print("value: " + str(dict['value']))
        print('-----')
        print("time: " + str(dict['time']))
        print('-----')


UDIT = Client()
UGC = Client()
AICTE = Client()
MU = Client()

t1 = Transaction(UDIT, UGC.identity, 15.0)
t1.sign_transaction()
transactions = [t1]
t2 = Transaction(UDIT, AICTE.identity, 6.0)
t2.sign_transaction()
transactions.append(t2)
t3 = Transaction(UGC, MU.identity, 2.0)
t3.sign_transaction()
transactions.append(t3)
t4 = Transaction(AICTE, UGC.identity, 4.0)
t4.sign_transaction()
transactions.append(t4)
for transaction in transactions:
    Transaction.display_transaction(transaction)
    print(" ")
```

Output :

```
C:\Users\murarilal\Desktop\Blockchain\ethermine>python Practical1C.py
sender: 30819f300d06092a864886f70d01010105000381 8d00308189028181009e90a987d58c9d3f6c6648b3bc7257b27a5f9be39c4ea789e26
018716b2f4aa52ebb16a6c6e5e18fb68b86d9f337b893c26f0d8979a43244a9c1bb36cbcc05fc7bab65e4edeacc58ffdb85389240cf264e5e2dfd
cb62ce19f248ed674578ecbc06d5005fafe0eae55a470bdb77e76e5f76495175a8d067714bd2349deb60aae30203010001
-----
recipient: 30819f300d06092a864886f70d01010105000381 8d00308189028181009d6d2d6e755a03e140ecf8cbb1de6baef687c35fc312392d
4df651e330daf802e2df34258ec8ae8a06919d49ea0feb4eeed8ef4abb98bd6433a6525231ced53df53d09aa9c5c955471d5dabb526f8885b60ee
820d6b8b7991abb881c6bdddaa9339c3ad413f1804d04e63fc40da2180c20992d7ae9eb4f4273f6d2ad4bc06dcb0203010001
-----
value: 15.0
-----
time: 2024-08-07 19:27:52.091886
-----
```

```
sender: 30819f300d06092a864886f70d01010105000381 8d00308189028181009e90a987d58c9d3f6c6648b3bc7257b27a5f9be39c4ea789e26018716
b2f4aa52ebb16a6c6e5e18fb68b86d9f337b893c26f0d8979a43244a9c1bb36cbcc05fc7bab65e4edeacc58ffdb85389240cf264e5e2dfdcb62ce19f248
ed674578ecbc06d5005fafe0eae55a470bdb77e76e5f76495175a8d067714bd2349deb60aae30203010001
-----
recipient: 30819f300d06092a864886f70d01010105000381 8d00308189028181009c1302ebe2f47aa9d6176856c6d11717d4d95cc98572d30daab6d25
5119afca6d261c407aef670d62a9227aef8c9a10eebd6f446881628917ebc4ec0e4cb5470cb8e04dca290e3d122ff1a5be110a4da5b53e21df72f27fb05
d9883cabf16a70bda71b04dc540a8eec05e19f2158591196392d6c1ef2dd7f207999b2f130e94230203010001
-----
value: 6.0
-----
time: 2024-08-07 19:27:52.107511
-----
```

```
sender: 30819f300d06092a864886f70d01010105000381 8d00308189028181009d6d2d6e755a03e140ecf8cbb1de6baef687c35fc312392d4df651e33
0daf802e2df34258ec8ae8a06919d49ea0feb4eeed8ef4abb98bd6433a6525231ced53df53d09aa9c5c955471d5dabb526f8885b60ee820d6b8b7991abb
881c6bdddaa9339c3ad413f1804d04e63fc40da2180c20992d7ae9eb4f4273f6d2ad4bc06dcb0203010001
-----
recipient: 30819f300d06092a864886f70d01010105000381 8d00308189028181009b74527586b394d6f32bf33e2f6cbc53de196f7e68a5fc5e602fe63
614749c8eccf31b933e7df8538d56ae0c2a27fb829dbcd5ebf0a43fec3b20578ff510b39bb9585efff1a28b6073c223faa1122cac8da70a8465a4f947f5
e41a95bb83f24011e0a191b9b3f958bc58872bed4af8493fed107bdf44d05badf984e45745dbbd90203010001
-----
value: 2.0
-----
time: 2024-08-07 19:27:52.107511
-----
```

```
sender: 30819f300d06092a864886f70d01010105000381 8d00308189028181009c1302ebe2f47aa9d6176856c6d11717d4d95cc98572d30daab6d25511
9afca6d261c407aef670d62a9227aef8c9a10eebd6f446881628917ebc4ec0e4cb5470cb8e04dca290e3d122ff1a5be110a4da5b53e21df72f27fb05d98
83cabf16a70bda71b04dc540a8eec05e19f2158591196392d6c1ef2dd7f207999b2f130e94230203010001
-----
recipient: 30819f300d06092a864886f70d01010105000381 8d00308189028181009d6d2d6e755a03e140ecf8cbb1de6baef687c35fc312392d4df651
e330daf802e2df34258ec8ae8a06919d49ea0feb4eeed8ef4abb98bd6433a6525231ced53df53d09aa9c5c955471d5dabb526f8885b60ee820d6b8b7991
abb881c6bdddaa9339c3ad413f1804d04e63fc40da2180c20992d7ae9eb4f4273f6d2ad4bc06dcb0203010001
-----
value: 4.0
-----
time: 2024-08-07 19:27:52.123134
-----
```

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

Code :

```
#Practical1 D
import binascii
import collections
import datetime
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto import Random
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("asci
i")
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):
        identity = "Genesis" if self.sender == "Genesis" else
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")

    def display_transaction(transaction):
```

```
    # for transaction in transactions:
        dict = transaction.to_dict()
        print("sender: " + dict['sender'])
        print('-----')
        print("recipient: " + dict['recipient'])
        print('-----')
        print("value: " + str(dict['value']))
        print('-----')
        print("time: " + str(dict['time']))
        print('-----')
class Block:
    def __init__(self, client):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""
        self.client = client
def dump_blockchain(blocks):
    print(f"\nNumber of blocks in the chain: {len(blocks)}")
    for i, block in enumerate(blocks):
        print(f"block # {i}")
        for transaction in block.verified_transactions:
            Transaction.display_transaction(transaction)
            print(" ")
    print(" ")
UDIT = Client()
t0 = Transaction("Genesis", UDIT.identity, 500.0)
block0 = Block(UDIT)
block0.previous_block_hash = ""
NONCE = None
block0.verified_transactions.append(t0)
digest = hash(block0)
last_block_hash = digest
TPCoins = [block0]
dump_blockchain(TPCoins)
```

Output :

```
C:\Users\murarilal\Desktop\Blockchain\ethermine>python Practical1D.py

Number of blocks in the chain: 1
block # 0
sender: Genesis
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b009553808c3ec3bc2a741c056ed00e3ab0e36d3b629caf9a77c72
bcbf196bf7508f30641112d26d9a604e3ce1a41445c8fd1bcd000350af7194f4d81cf4fbfd55b309d60c8b244101359f7680fbbf42a1c7b0bec803d20b3
988867476bc3fc33f1145e4ed0cf0ad614d54911fd886d73fc00901b80f774252e946e5a491e1350203010001
-----
value: 500.0
-----
time: 2024-08-07 19:37:49.060660
-----
```

[E] Create a mining function and test it.

Code :

```
#Practical1 E
import hashlib


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


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

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

Output :

```
C:\Users\murarilal\Desktop\Blockchain\ethermine>python Practical1E.py
After 285 iterations found nonce: 1101ec9ad00e69e3df2f94e952e59f0358e9bdccbe35a78fcc722d1cdf912954
1101ec9ad00e69e3df2f94e952e59f0358e9bdccbe35a78fcc722d1cdf912954
```

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

Code :

```
#Practical1 F
import datetime
import hashlib
class Block:
  def __init__(self, data, previous_hash):
    self.timestamp = datetime.datetime.now(datetime.timezone.utc)
    self.data = data
    self.previous_hash = previous_hash
    self.hash = self.calc_hash()
  def calc_hash(self):
    sha = hashlib.sha256()
    hash_str = self.data.encode("utf-8")
    sha.update(hash_str)
    return sha.hexdigest()
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 :

```
C:\Users\murarilal\Desktop\Blockchain\ethermine>python Practical1F.py
Timestamp: 2024-08-07 14:17:52.148699+00:00
Data: First block
Previous Hash: 0
Hash: 876fb923a443ba6afe5fb32dd79961e85be2b582cf74c233842b630ae16fe4d9

Timestamp: 2024-08-07 14:17:52.148699+00:00
Data: Second block
Previous Hash: 876fb923a443ba6afe5fb32dd79961e85be2b582cf74c233842b630ae16fe4d9
Hash: 8e2fb9e02898feb024dff05ee0b27fd5ea0a448e252d975e6ec5f7b0a252a6cd

Timestamp: 2024-08-07 14:17:52.148699+00:00
Data: Third block
Previous Hash: 8e2fb9e02898feb024dff05ee0b27fd5ea0a448e252d975e6ec5f7b0a252a6cd
Hash: 06e369fbfbe5362a8115a5c6f3e2d3ec7292cc4272052dcc3280898e3206208d
```

PRACTICAL 2

AIM – Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.

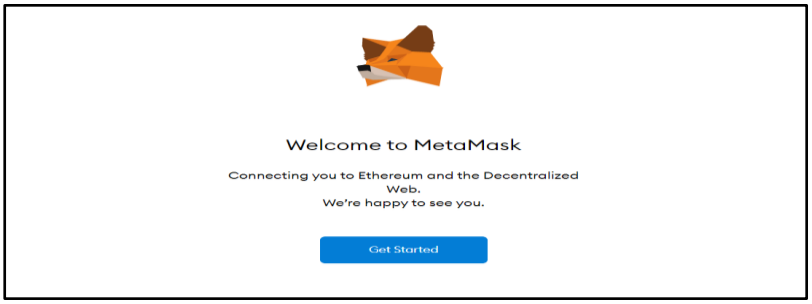
Step 1: Go to Chrome Web Store Extensions Section.

Step 2: Search MetaMask.

Step 3: Check the number of downloads to make sure that the legitimate MetaMask is being installed, as hackers might try to make clones of it.

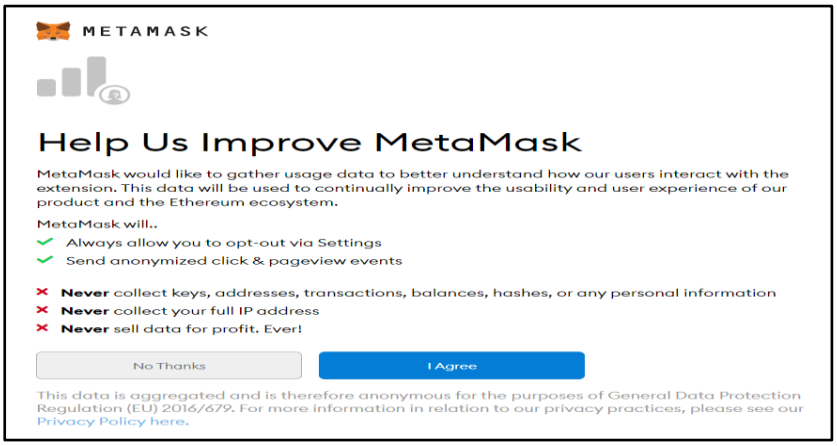
Step 4: Click the Add to Chrome button.

Step 5: Once installation is complete this page will be displayed. Click on the Get Started button.

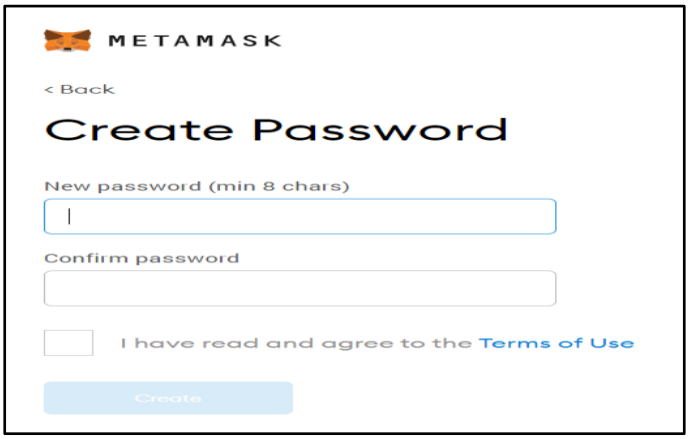


Step 6: This is the first time creating a wallet, so click the Create a Wallet button. If there is already a wallet then import the already created using the Import Wallet button.

Step 7: Click I Agree button to allow data to be collected to help improve MetaMask or else click the No Thanks button. The wallet can still be created even if the user will click on the No thanks Button.

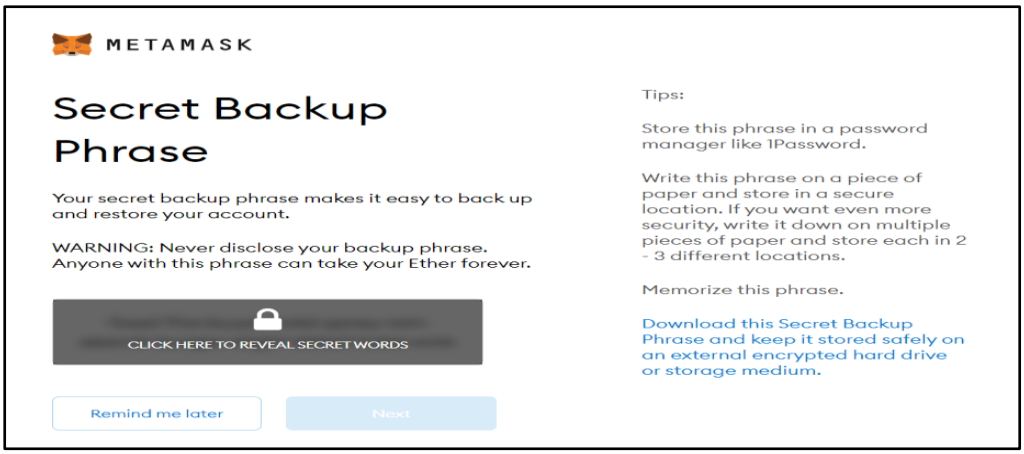


Step 8: Create a password for your wallet. This password is to be entered every time the browser is launched and wants to use MetaMask. A new password needs to be created if chrome is uninstalled or if there is a switching of browsers. In that case, go through the Import Wallet button. This is because MetaMask stores the keys in the browser. Agree to Terms of Use.

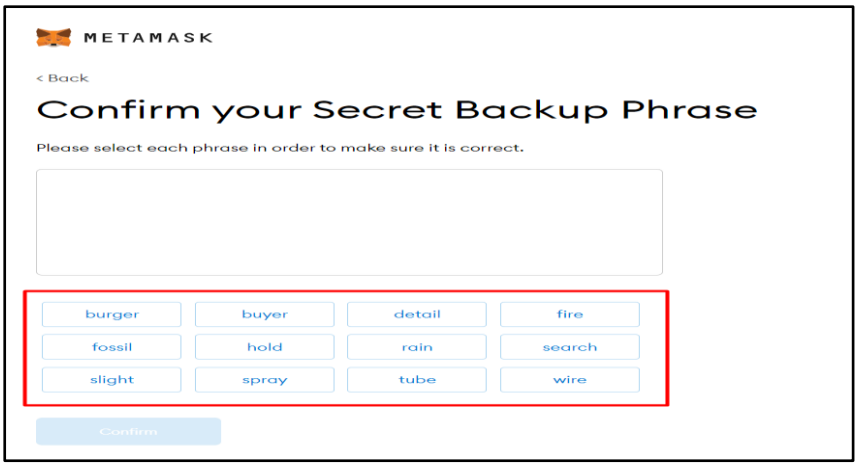


Step 9: Click on the dark area which says Click here to reveal secret words to get your secret phrase.

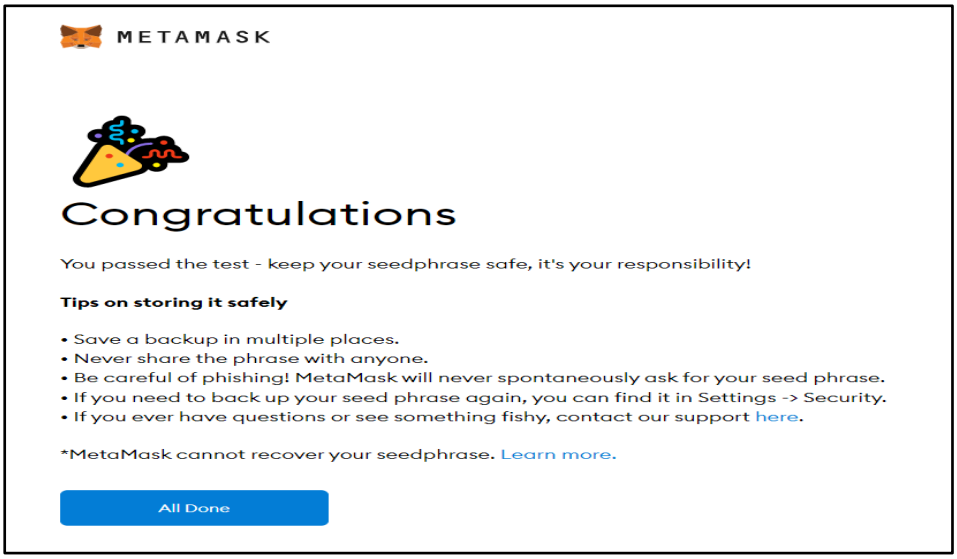
Step 10: This is the most important step. Back up your secret phrase properly. Do not store your secret phrase on your computer. Please read everything on this screen until you understand it completely before proceeding. The secret phrase is the only way to access your wallet if you forget your password. Once done click the Next button.



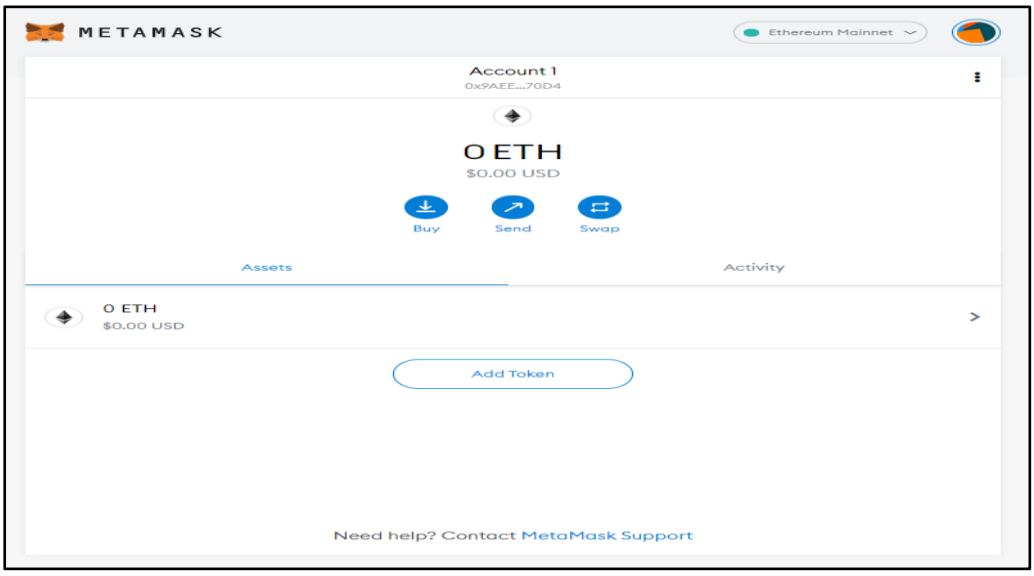
Step 11: Click the buttons respective to the order of the words in your seed phrase. In other words, type the seed phrase using the button on the screen. If done correctly the Confirm button should turn blue.



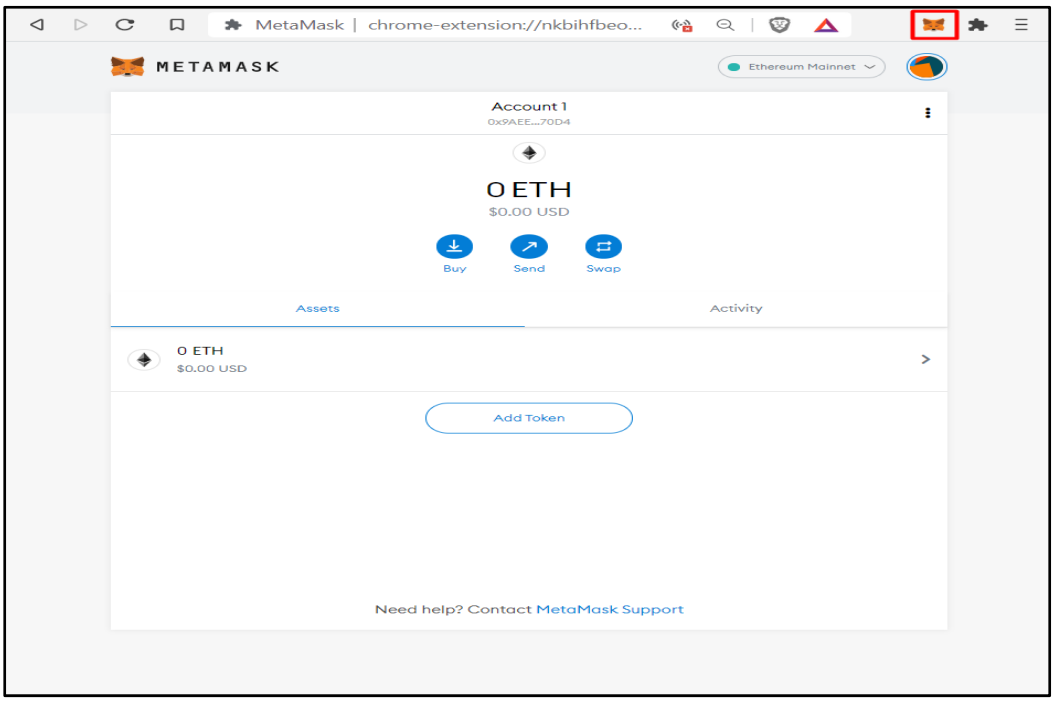
Step 12: Click the Confirm button. Please follow the tips mentioned.



Step 13: One can see the balance and copy the address of the account by clicking on the Account 1 area.



Step 14: One can access MetaMask in the browser by clicking the Foxface icon on the top right. If the Foxface icon is not visible, then click on the puzzle piece icon right next to it.



PRACTICAL 3

AIM - Implement and demonstrate the use of the following in Solidity:

[A] Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables.

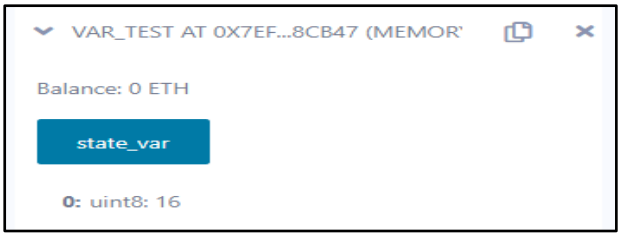
[I] Variable

[i] State Variable

CODE:

```
//Solidity program to demonstrate state variables
pragma solidity ^0.8.26;
// Creating a contract
contract var_Test
{
   // Declaring a state variable
   uint8 public state_var;

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

OUTPUT:



[ii] Local Variable

CODE:

```
//Solidity program to demonstrate Local variables
pragma solidity ^0.5.0;
// Creating a contract
contract local_var_Test
{
```

```
    // Defining function to show the declaration and
    // scope of Local variables
    function acsess_local_variable() public pure returns(uint) {
        // Initializing Local variables
        uint a = 10;
        uint b = 40;
        uint sum = a + b;
        // Access the Local variable
        return sum;
    }
}
```

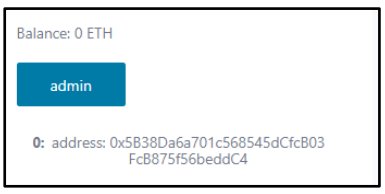
OUTPUT:

[iii] Global Variable

CODE:

```
//Solidity program to show Global variables
pragma solidity ^0.8.26;
contract globalTest
{
    // Defining a variable
    address public admin;
    // Creating a constructor to
    // use Global variable
    constructor() public
    {
        admin = msg.sender;
    }
}
```

OUTPUT:



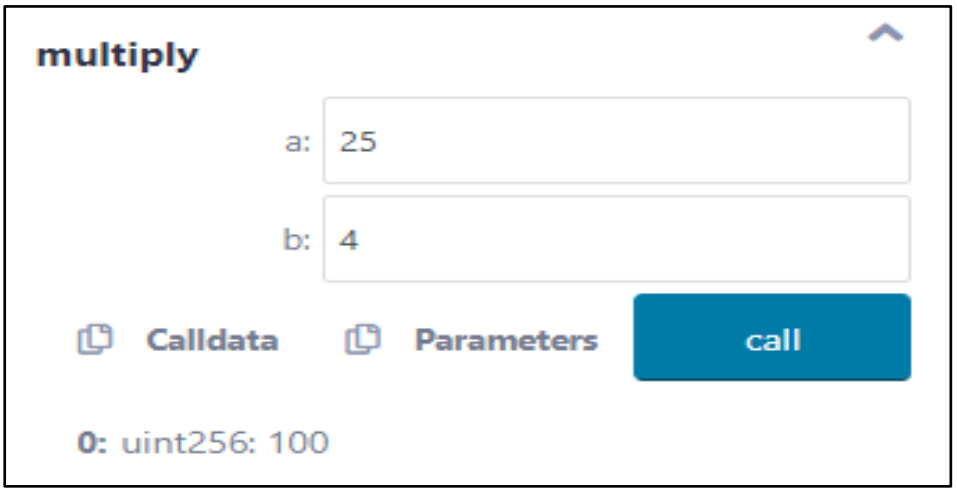
[II] Operators

CODE:

```
pragma solidity ^0.8.26;

contract Operators {

    uint256 result = 0;

    function addition(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }

    function subtraction(uint256 a, uint256 b) public pure returns (uint256) {
        return a - b;
    }

    function division(uint256 a, uint256 b) public pure returns (uint256) {
        return a / b;
    }

    function multiply(uint256 a, uint256 b) public pure returns (uint256) {
        return a * b;
    }
}
```

OUTPUT:


division
a: 15
b: 3
Calldata
Parameters
call
0: uint256: 5


multiply
a: 25
b: 4
Calldata
Parameters
call
0: uint256: 100

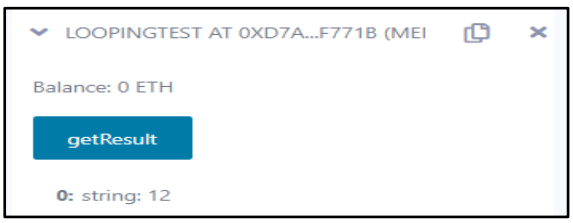

subtraction
a: 23
b: 12
Calldata
Parameters
call
0: uint256: 11

[III] Loops

CODE:

```
pragma solidity ^0.8.26;
contract LoopingTest {
    uint256 storedData;
    constructor() public {
        storedData = 10;
    }
    function getResult() public pure returns (string memory) {
        uint256 a = 10;
        uint256 b = 2;
        uint256 result = a + b;
        return integerToString(result);
    }
    function integerToString(uint256 _i) internal pure returns (string memory) {
        if (_i == 0) {
            return "0";
        }
        uint256 j = 0;
        uint256 len;
        for (j = _i; j != 0; j /= 10) {
            //for loop example
            len++;
        }
        bytes memory bstr = new bytes(len);
        uint256 k = len - 1;
        while (_i != 0) {
            bstr[k--] = bytes1(uint8(48 + (_i % 10)));
            _i /= 10;
        }
        return string(bstr); //access local variable
    }
}
```

OUTPUT:



[IV] Decision Making

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

// Creating a contract
contract Types {

    // Declaring state variables
    uint256 i = 10;
    string result;

    function decision_making() public payable returns (string memory) {
        if (i < 10) {
            result = "less than 10";
        }
        else if (i == 10) {
            result = "equal to 10";
        }
        else {
            result = "greater than 10";
        }
        return result;
    }
}
```

OUTPUT:

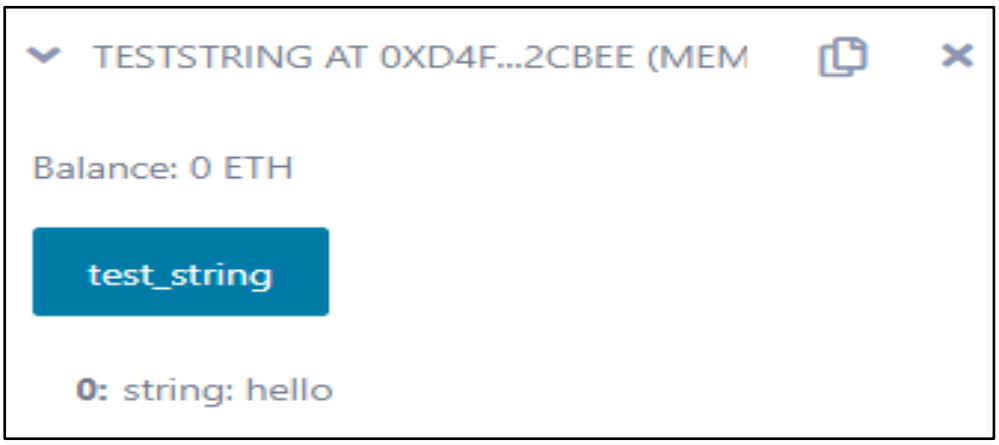
[V] Strings

[i] Double Quotes

CODE:

```
pragma solidity ^0.8.26;

contract testString
{
   function test_string() public pure returns (string memory)
   {
      string memory a = "hello";
      return a;
   }
}
```

OUTPUT:



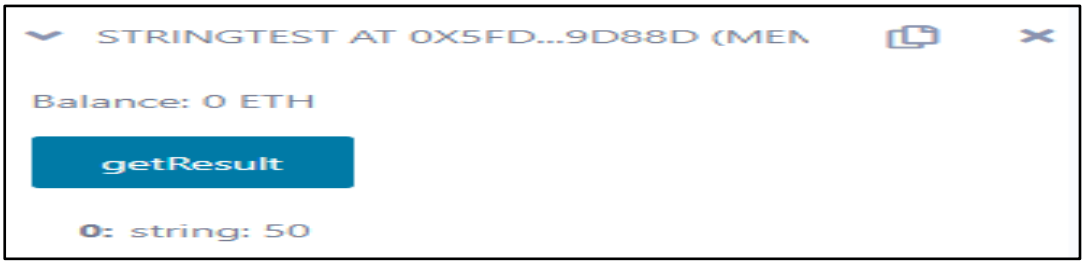
[ii] Single Quotes

CODE:

```
pragma solidity ^0.8.26;
contract stringTest
{
   function getResult() public pure returns(string memory) {
      uint a = 25;
      uint b = 25;
      uint result = a + b;
      return integerToString(result);
   }
   function integerToString(uint _i) internal pure returns (string memory){
```

```
        if (_i == 0)
        {
            return "0";
        }
        uint j = _i;
        uint len;
        while (j != 0)
        {
            len++;
            j/= 10;
        }
        bytes memory bstr = new bytes(len);
        uint k = len - 1;
        while (_i != 0)
        {
            bstr[k-- ] = byte(uint8(48 + _i % 10));
            _i /= 10;
        }
    return string(bstr);
    }
}
```

OUTPUT:

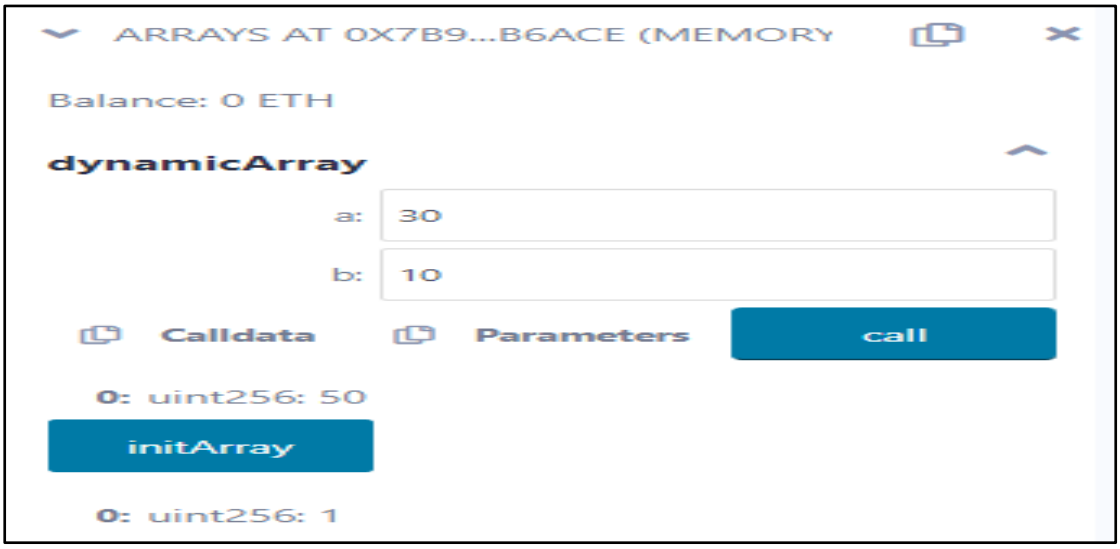


[VI] Arrays

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;

contract Arrays {

    function initArray() public pure returns (uint256) {
        uint128[3] memory array = [1, 2, uint128(3)];
        return array[0];
    }

    function dynamicArray(uint256 a, uint256 b) public pure returns (uint256) {

        uint128[] memory array = new uint128[](a);
        uint128 val = 5;

        for (uint128 j = 0; j < a; j++) {
            array[j] = j * val;
        }

        return array[b];
    }
}
```

OUTPUT:

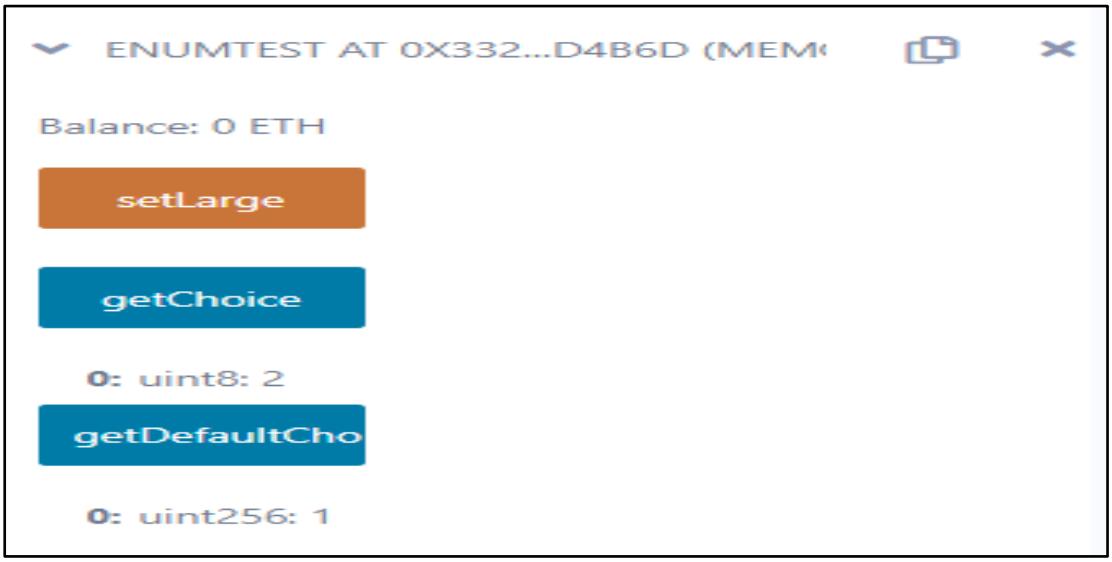


[VII] Enums

CODE:

```
pragma solidity ^0.8.26;

contract enumTest {
    enum FreshJuiceSize {
        SMALL,
        MEDIUM,
        LARGE
    }
    FreshJuiceSize choice;
    FreshJuiceSize constant defaultChoice = FreshJuiceSize.MEDIUM;

    function setLarge() public {
        choice = FreshJuiceSize.LARGE;
    }

    function getChoice() public view returns (FreshJuiceSize) {
        return choice;
    }

    function getDefaultChoice() public pure returns (uint256) {
        return uint256(defaultChoice);
    }
}
```

OUTPUT:

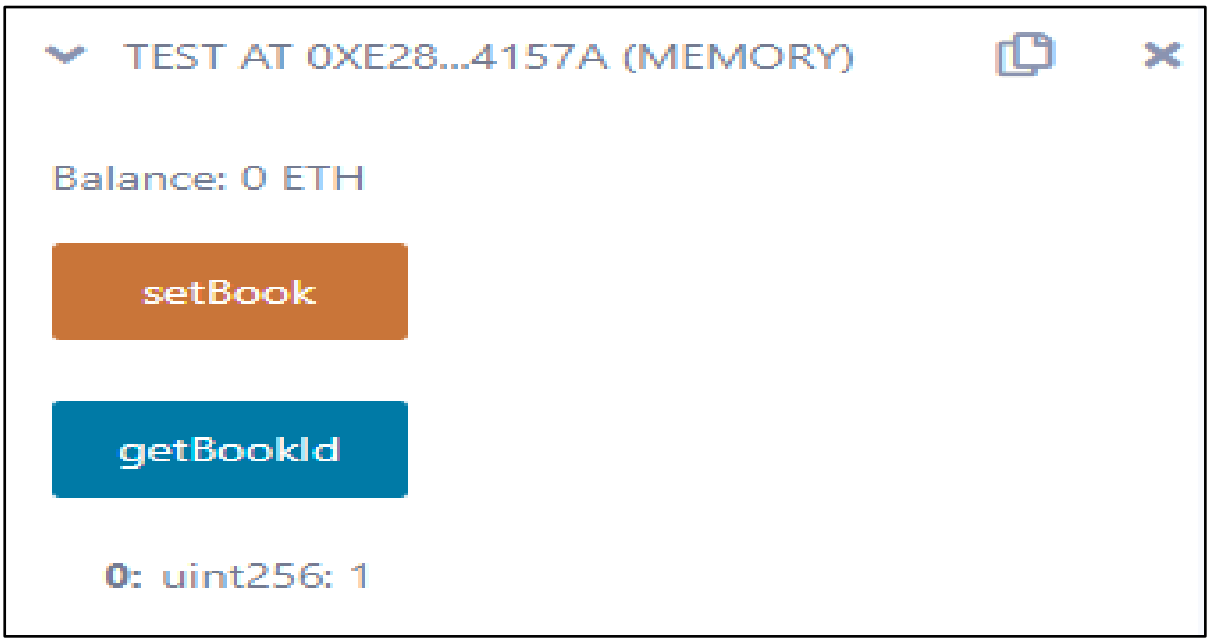


[VIII] Structs

STEPS:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

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:



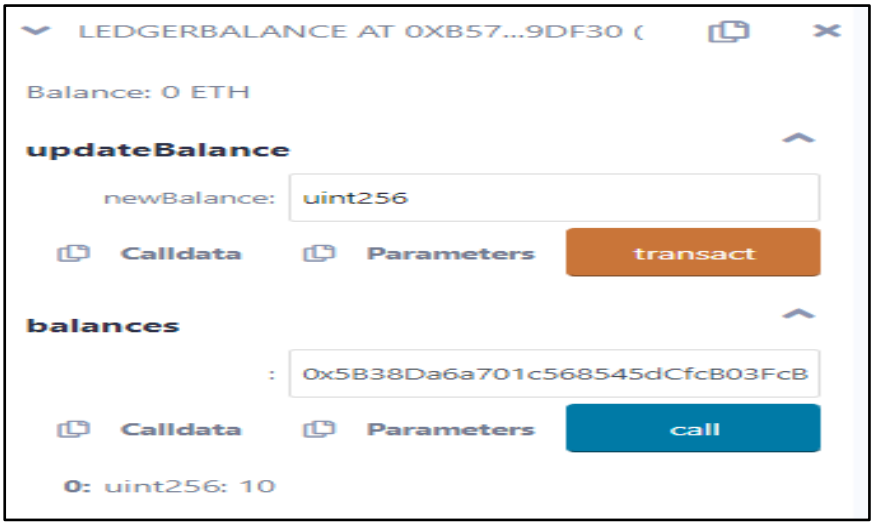
[IX] Mappings

CODE:

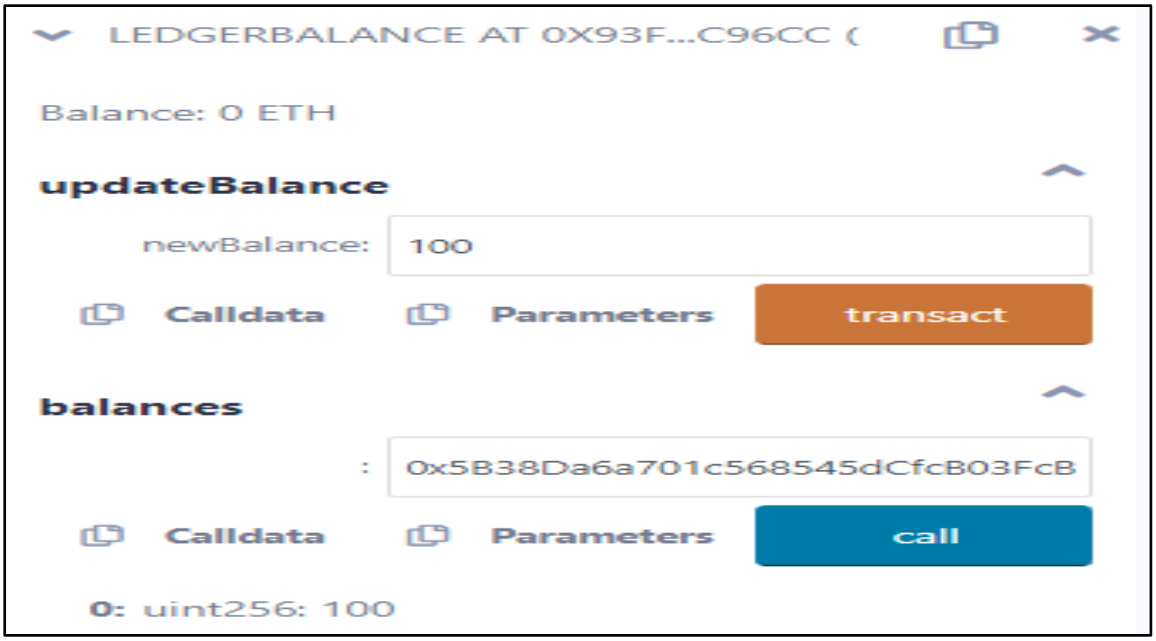
```
pragma solidity ^0.8.26;

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

   function updateBalance(uint newBalance) public
   {
   balances[msg.sender] = newBalance;
   }
}
contract Updater
{
   function updateBalance() public returns (uint)
   {
      LedgerBalance ledgerBalance = new LedgerBalance();
      ledgerBalance.updateBalance(10);
      return ledgerBalance.balances(address(this));
   }
}
```

OUTPUT:

With Original Balance: 10



With Updated Balance: 100

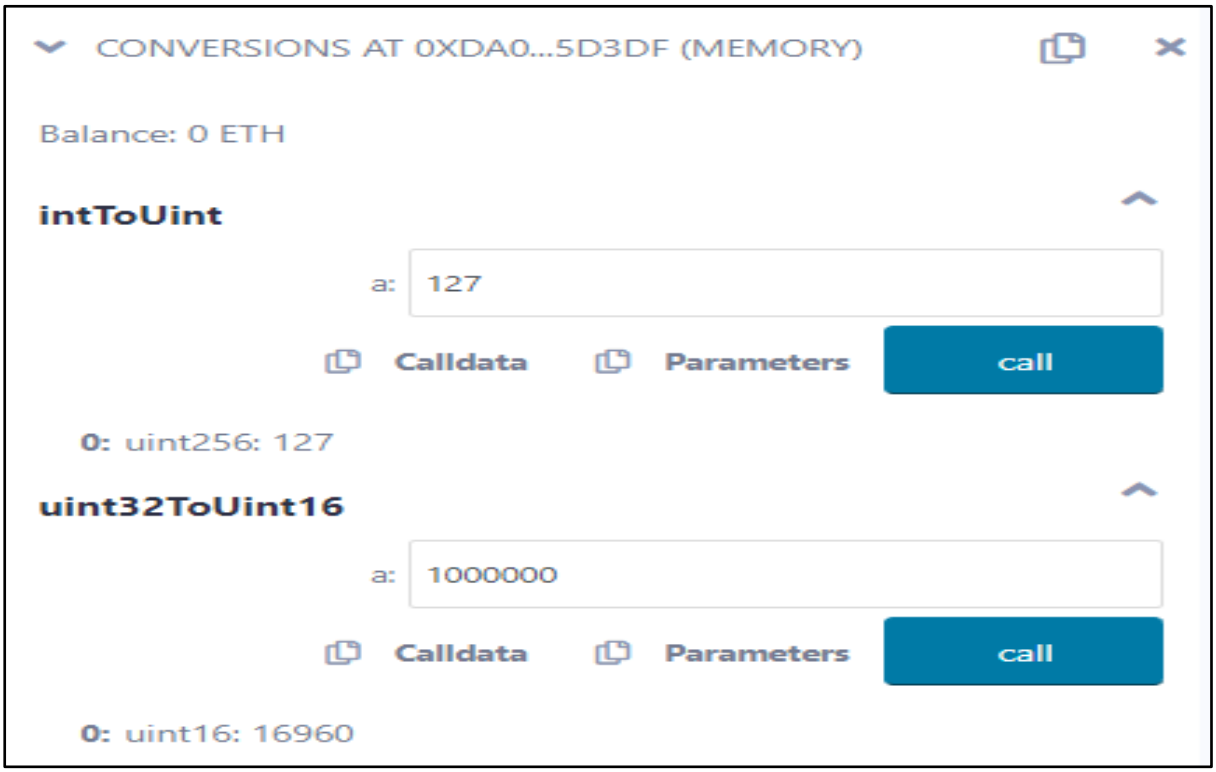


[X] Conversions

CODE:

```
pragma solidity ^0.8.26;

contract Conversions {
    function intToUint(int8 a) public pure returns (uint256) {
        uint256 b = uint256(a);
        return b;
    }

    function uint32ToUint16(uint32 a) public pure returns (uint16) {
        uint16 b = uint16(a);
        return b;
    }
}
```

OUTPUT:



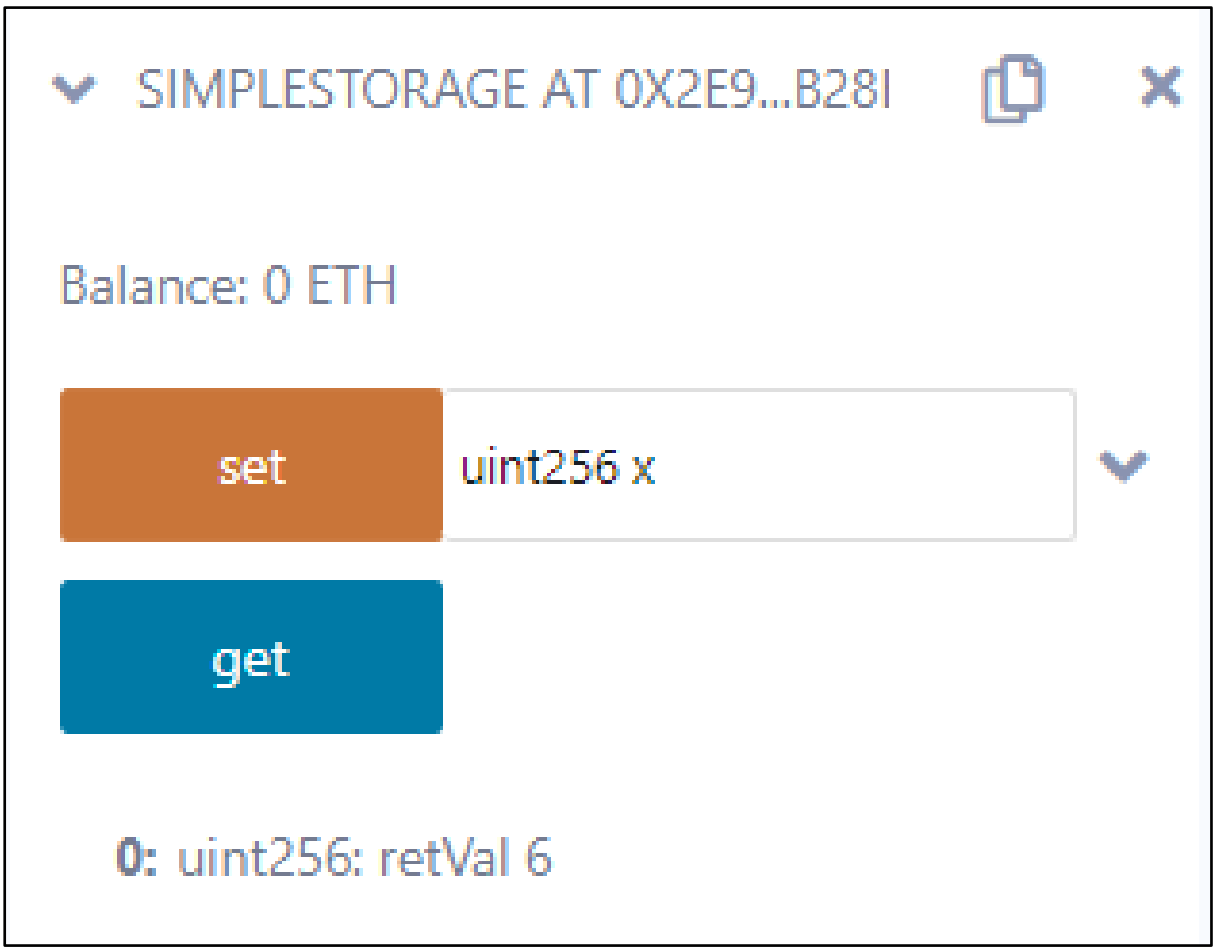
[XI] Ether Units

CODE:

```
pragma solidity ^0.8.26;

contract SimpleStorage {
    uint256 storedData = 2;

    function set(uint256 x) public {
        storedData = x;
        /*
        Ether Units
        Wei
        Finney
        Szabo
        Ether
        */
        if (2000000000000000000 == 2 ether) {
            storedData = 2;
        }
        else {
            storedData = 3;
        }
        /*
        Time Units
        seconds
        minutes
        hours
        days
        weeks
        month
        years
        */
        if (120 seconds == 2 minutes) {
            storedData = 6;
        }
        else {
            storedData = 9;
        }
    }
```

```
    function get() constant public returns (uint256 retVal)
    {
        return storedData;
    }
}
```

OUTPUT:

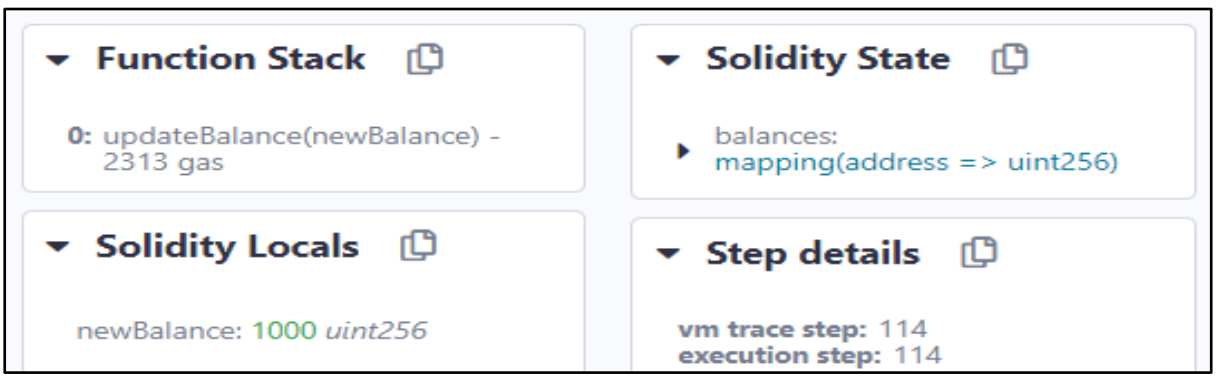


[XII] Special Variables

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

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

    function updateBalance(uint256 newBalance) public {
        balances[msg.sender] = newBalance;
    }
}
contract Updater {
    function updateBalance() public returns (uint256) {
        LedgerBalance ledgerBalance = new LedgerBalance();
        ledgerBalance.updateBalance(10);
        return ledgerBalance.balances(address(this));
    }
}
```

OUTPUT:



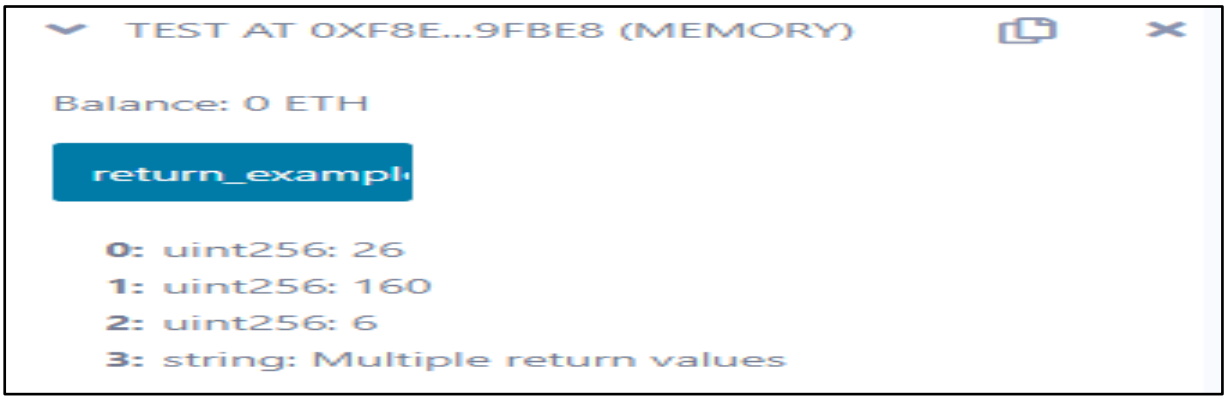
[B] Functions, Function Modifiers, View Functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions.

[I] Functions

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

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:

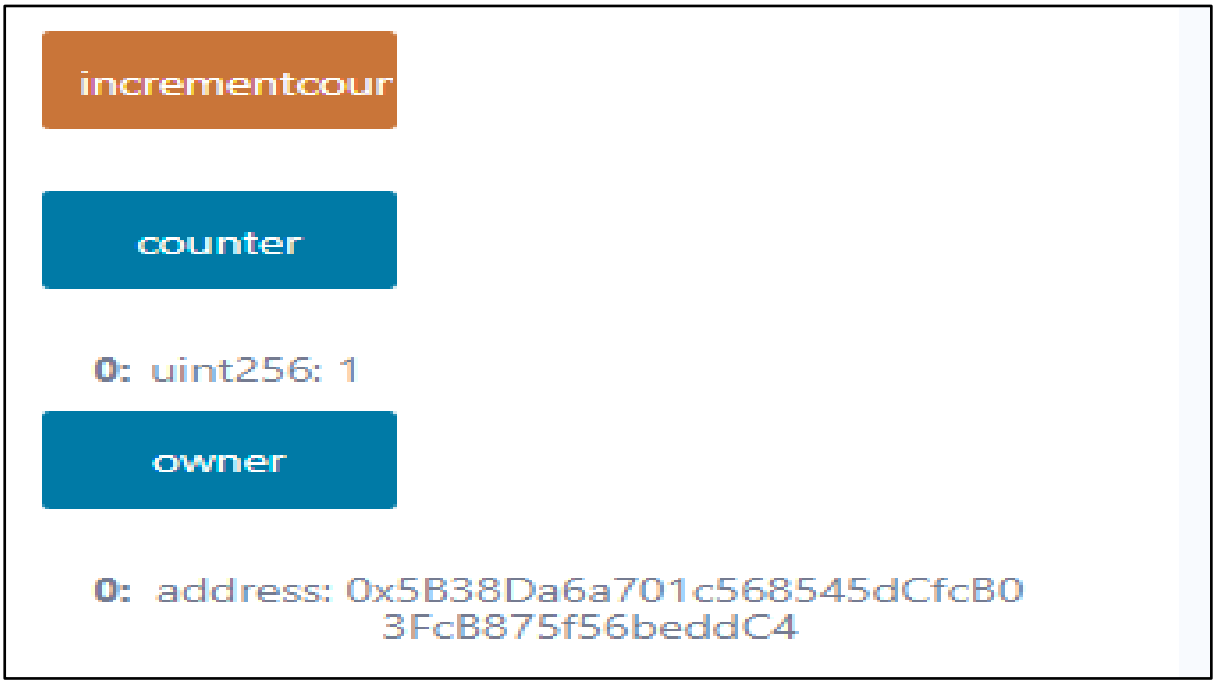


[II] Function Modifiers

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract ExampleContract {
  address public owner = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
  uint256 public counter;

  modifier onlyowner() {
    require(msg.sender == owner, "Only the contract owner can call");
    _;
  }

  function incrementcounter() public onlyowner {
    counter++;
  }
}
```

OUTPUT:

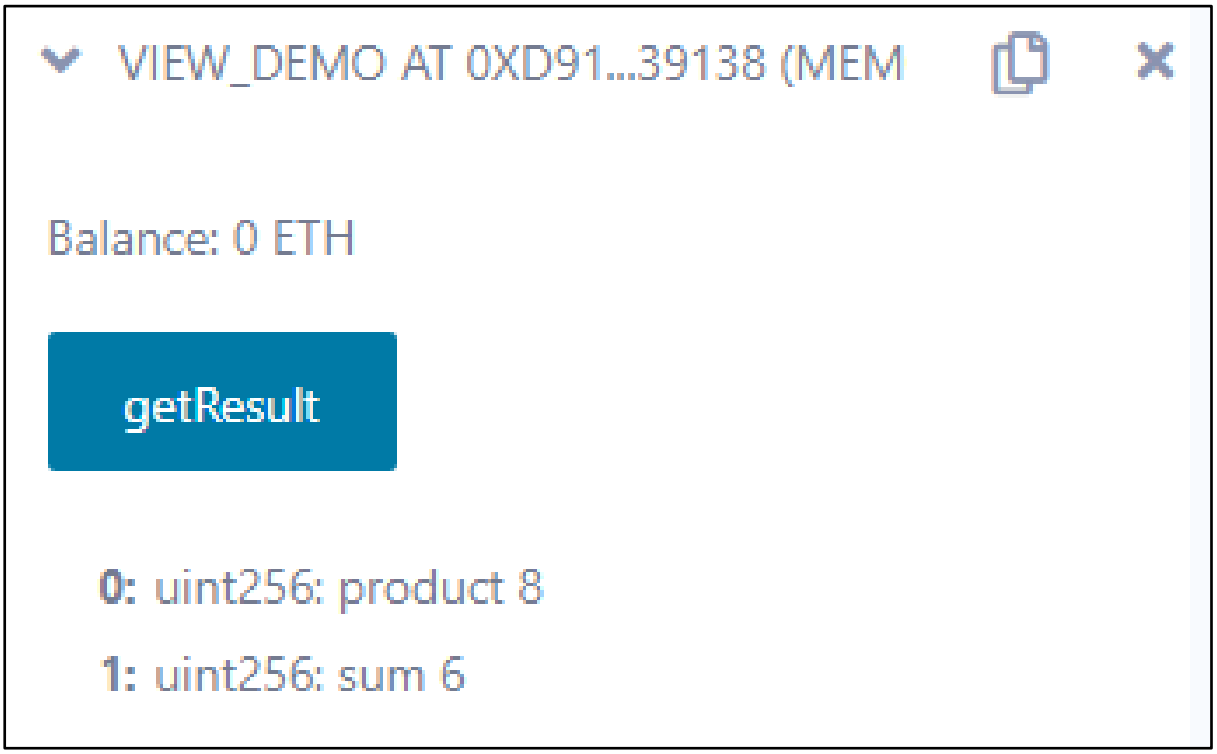


[III] View Functions

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

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

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

OUTPUT:

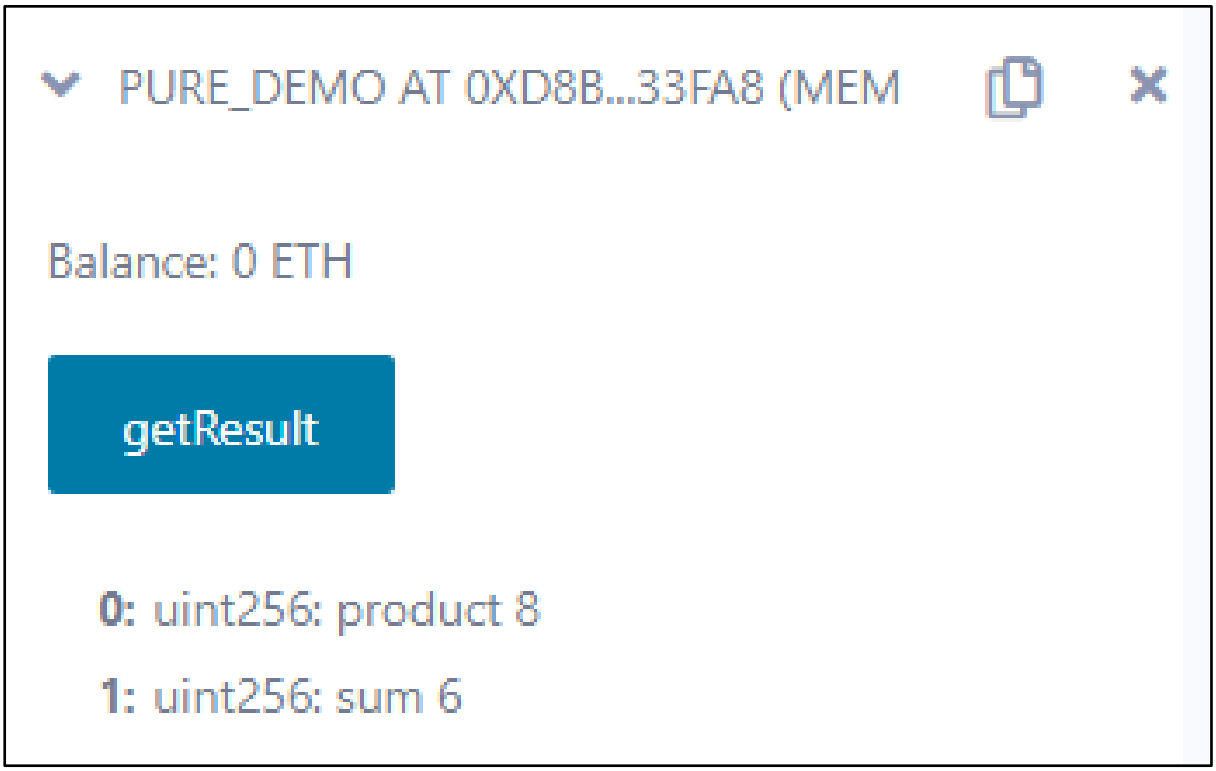


[IV] Pure Functions

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

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

OUTPUT:



[V] Fallback Function

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;
contract A {
  uint256 n;

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

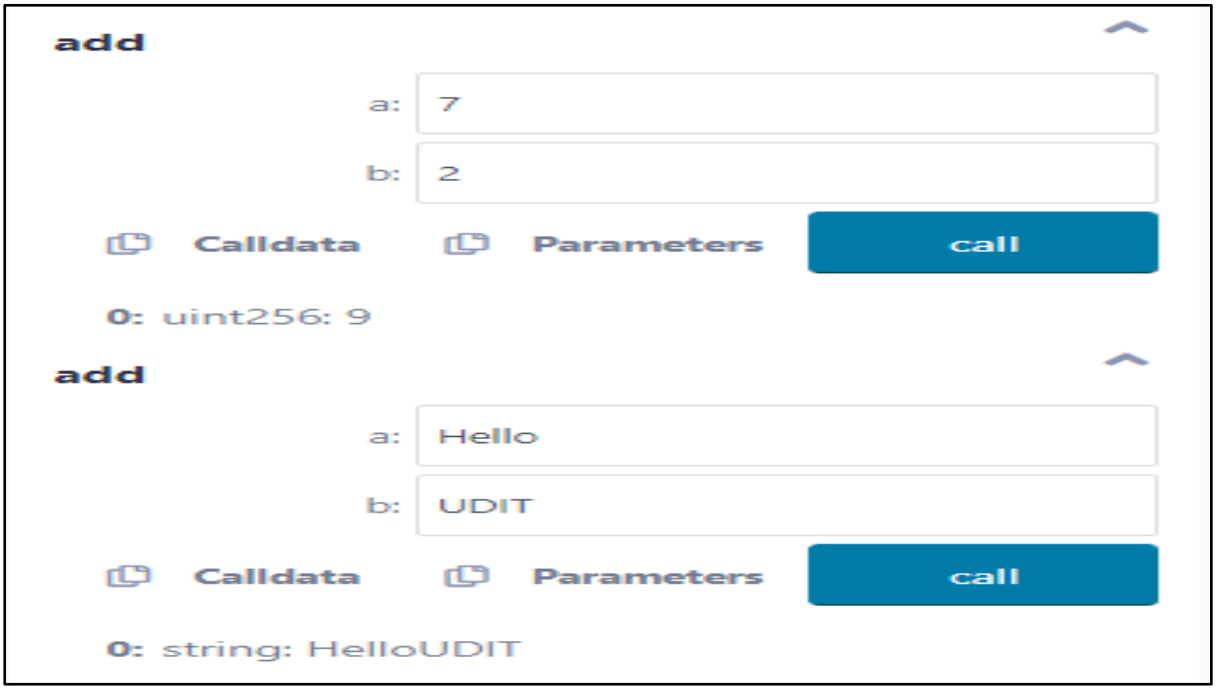
OUTPUT:

[VI] Function Overloading

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

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:

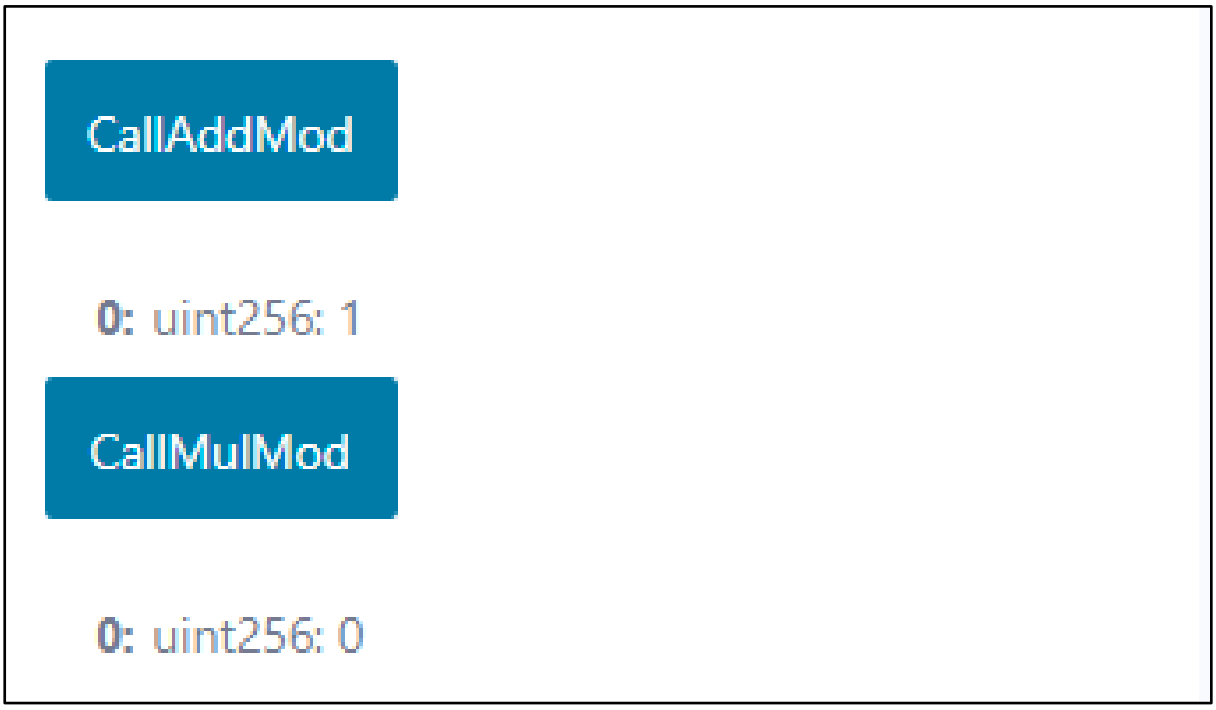


[VII] Mathematical Functions

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

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:



[VIII] Cryptographic Functions

CODE:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract Test{ function callKeccak256() public pure returns(bytes32 result){

  return keccak256("BLOCKCHAIN");

}

function callsha256() public pure returns(bytes32 result){

  return sha256("BLOCKCHAIN");

}

function callripemd() public pure returns (bytes20 result){

  return ripemd160("BLOCKCHAIN");
  }
}
```

OUTPUT:

PRACTICAL 4

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

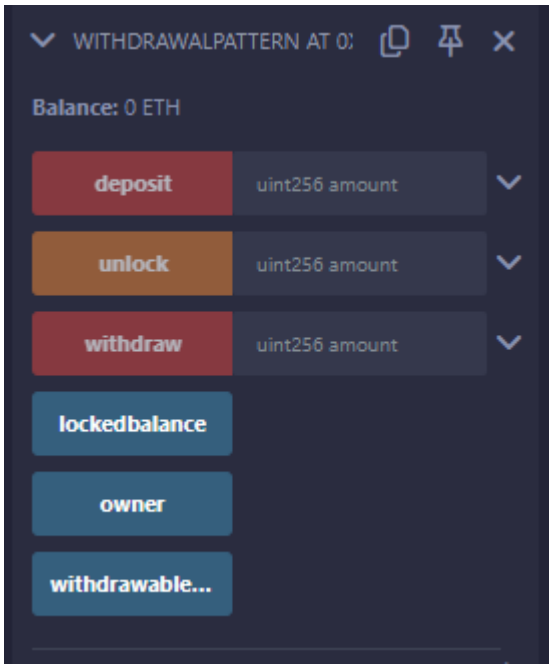
[A] Withdrawal Pattern, Restricted Access.

[I] Withdrawal Pattern.

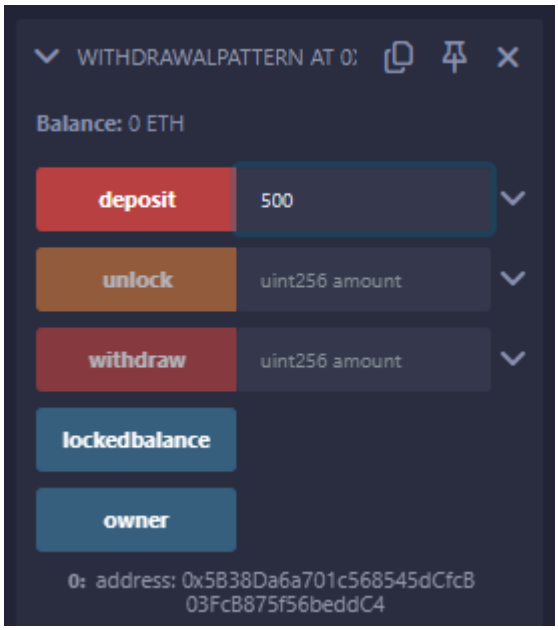
Code :

```
pragma solidity 0.8.26;

contract WithdrawalPattern {
    address public owner;
    uint256 public lockedbalance;
    uint256 public withdrawablebalance;

    constructor() {
        owner = msg.sender;
    }

  modifier onlyowner() {
        require(msg.sender == owner, "Only the owner can call this function");
        _;
    }

    function deposit(uint256 amount) public payable {
        require(amount > 0, "Amount must be greater than zero");
        lockedbalance += amount;
    }

    function withdraw(uint256 amount) public payable onlyowner {
        require(
            amount <= withdrawablebalance,
            "Insufficient withdrawable balance"
        );
     withdrawablebalance -= amount;
        payable(msg.sender).transfer(amount);
  }

    function unlock(uint256 amount) public onlyowner {
        require(amount <= lockedbalance, "Insufficient locked balance");
        lockedbalance -= amount;
        withdrawablebalance += amount;
    }
}
```

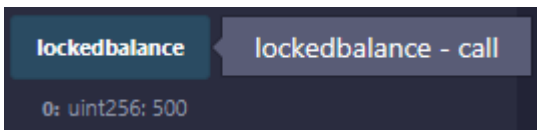
Output :



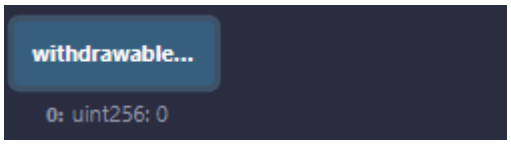
Step 1: Click on owner to create the owner object and add amount then click on deposit



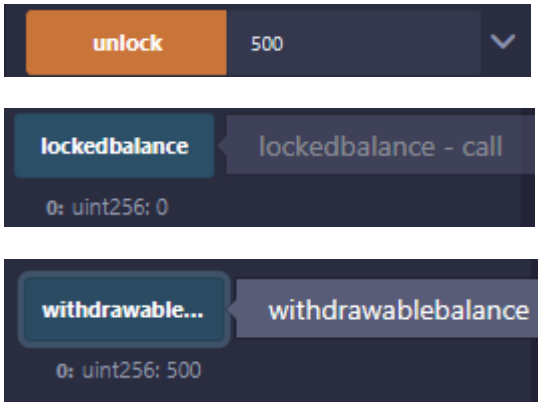
Step 2: Click on locked balance button to display the locked amount in the account.



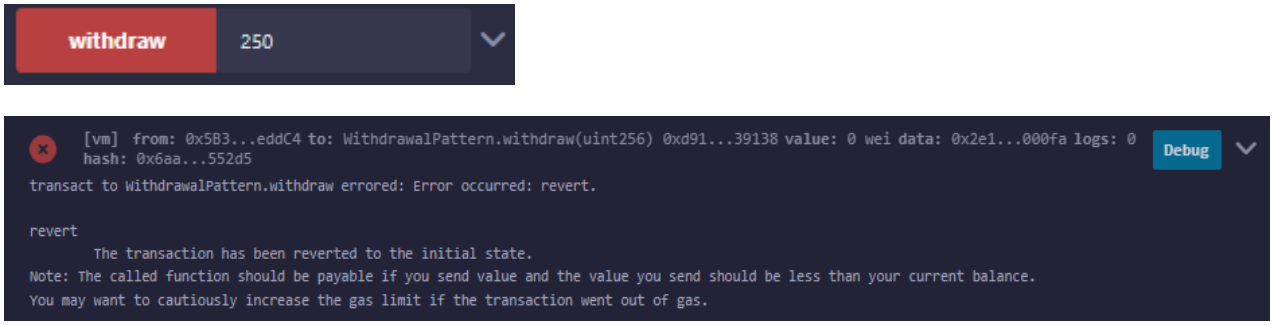
Step 3: Click on withdrawable balance button.



Step 4: Click on unlock button and enter any amount to transfer amount to withdrawable balance. Check locked balance and withdrawable balance.



Step 5: Enter any amount you want to withdraw and Click the withdraw button you should get an error and the transaction should be reverted.

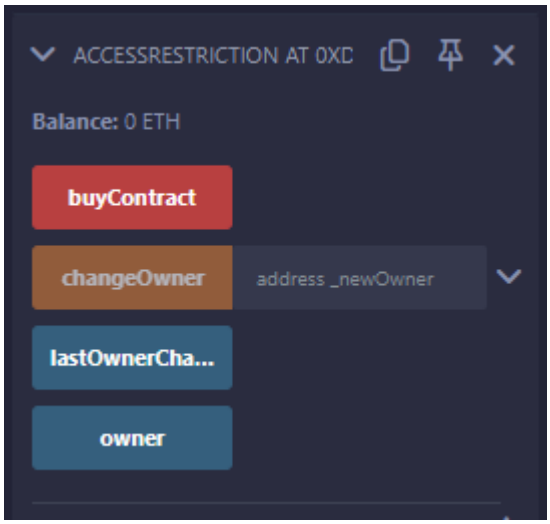


[II] Restricted Access.

Code :

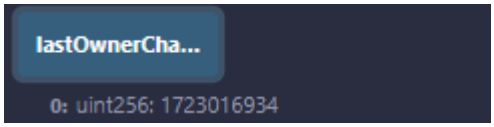
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract AccessRestriction {
    address public owner = msg.sender;

    uint256 public lastOwnerChange = block.timestamp;

    modifier onlyBy(address _account) {
        require(msg.sender == _account, "Not authorized");
        _;
    }

    modifier onlyAfter(uint256 _time) {
        require(block.timestamp >= _time, "Function called too early");
        _;
    }

    modifier costs(uint256 _amount) {
        require(msg.value >= _amount, "Insufficient value sent");
        _;
        if (msg.value > _amount) {
            (bool success, ) = msg.sender.call{value: msg.value - _amount}("");
            require(success, "Refund failed");
        }
    }

    function changeOwner(address _newOwner) public onlyBy(owner) {
        owner = _newOwner;
    }

    function buyContract()
        public
        payable
        onlyAfter(lastOwnerChange + 4 weeks)
        costs(1 ether)
    {
        owner = msg.sender;
        lastOwnerChange = block.timestamp;
    }
}
```

Output :

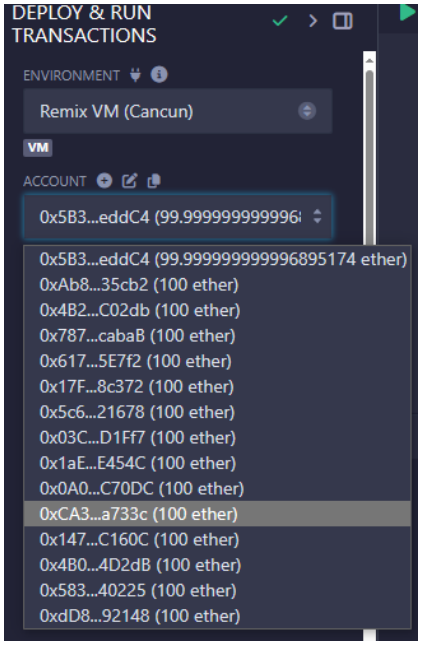


Step 1: Click on owner to create an owner object.

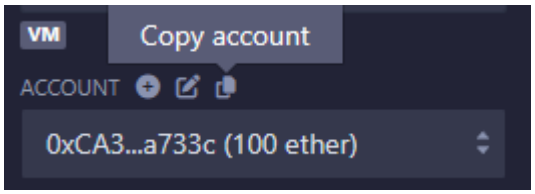
Step 2: Click on lastOwnerChange button.



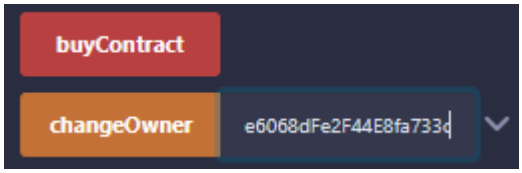
Step 3: Change the address of the account from Account dropdown in Deploy tab of Remix IDE.



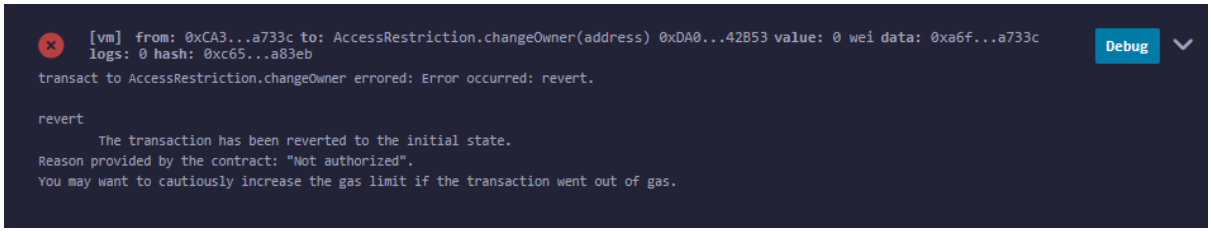
Step 4: Copy the address.



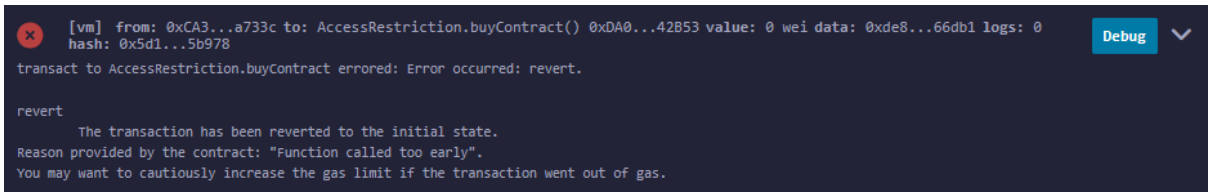
Step 5: Paste the address in changeOwner input and click on changeOwner.



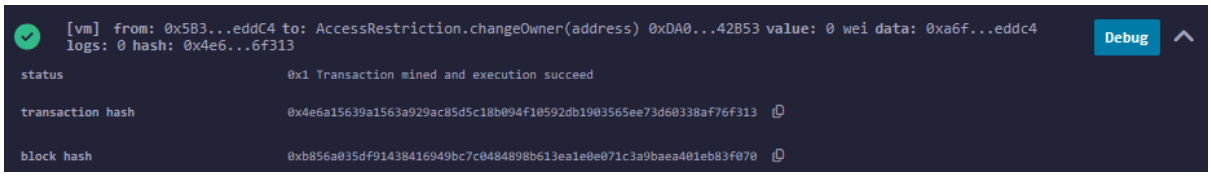
Step 6: You should get an error as following



Step 7: If you click on buycontract it should give an error as follows.



Step 8: Now, paste the actual address of the account in the changeowner input and click on changeowner.



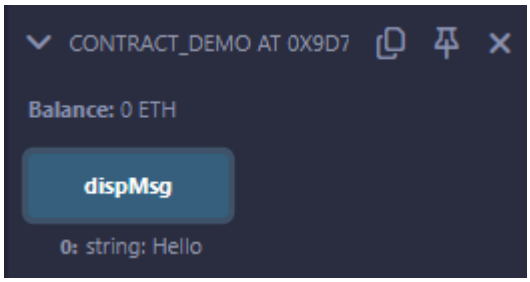
[B] Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces.

[I] Contracts

Code :

```
pragma solidity ^0.8.26;

contract Contract_demo {
   string message = "Hello";

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

Output :



[II] Inheritance

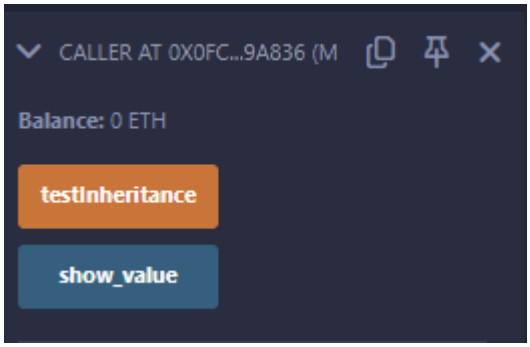
Code :

```
pragma solidity 0.8.26;
contract Parent {
   uint256 internal sum;
   function setValue() external {
      uint256 a = 10;
      uint256 b = 20;
      sum = a + b;
   }
}
contract child is Parent {
   function getValue() external view returns (uint256) {
      return sum;
   }
}
contract caller {
   child cc = new child();
   function testInheritance() public returns (uint256) {
      cc.setValue();
      return cc.getValue();
   }
   function show_value() public view returns (uint256) {
      return cc.getValue();
```

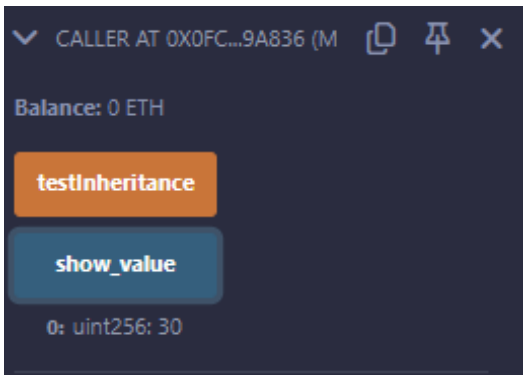
```
    }
}
```

Output :

Step 1: Select caller contract to deploy in Contract and deploy.



Step 2: Click test Inheritance and then click on show_value to view value.

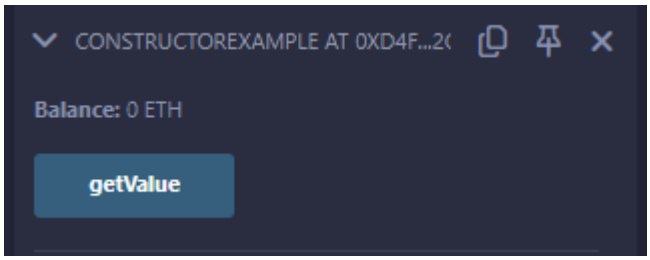


[III] Constructors

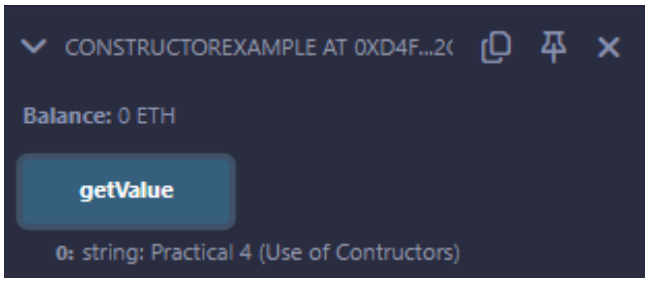
Code :

```
pragma solidity ^0.8.26;

// Creating a contract
contract constructorExample {
    string str;

    constructor() public {
        str = "Practical 4 (Use of Contructors)";
    }

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

Output :



Step 1: Click on getValue to print string



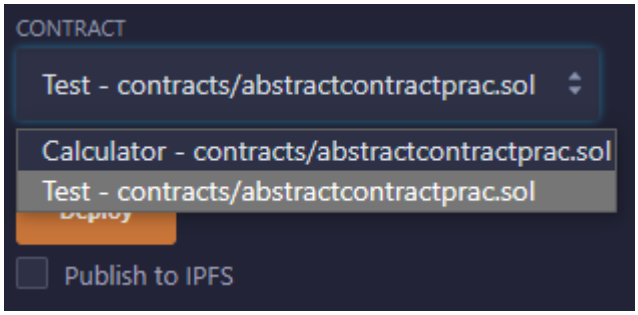
[IV] Abstract Contracts

Code :

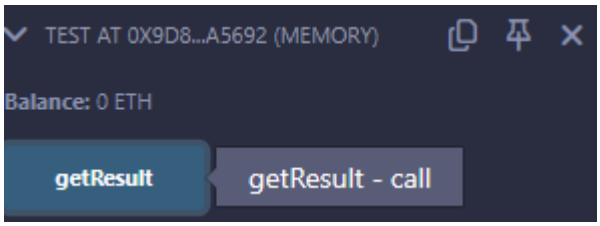
```
pragma solidity ^0.8.26;

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

contract Test is Calculator {
   constructor() {}

   function getResult() external view override returns (uint256) {
      uint256 a = 15;
      uint256 b = 20;
      uint256 result = a + b;
      return result;
   }
}
```

Output :

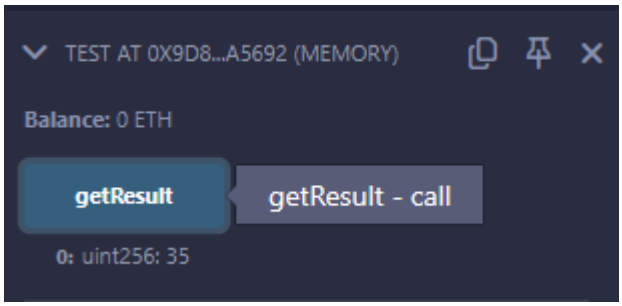
Step 1: Select Test contract and deploy.



Step 2: The contact will deploy as below.



Step 3: Click on getResult to get sum of a+b.



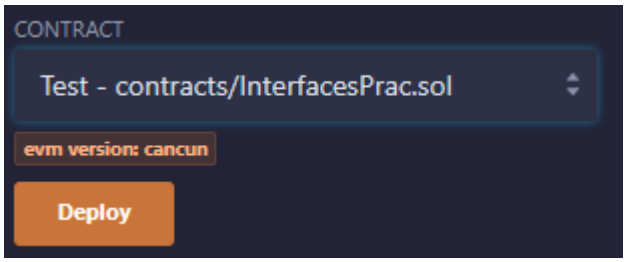
[V] Interfaces

Code :

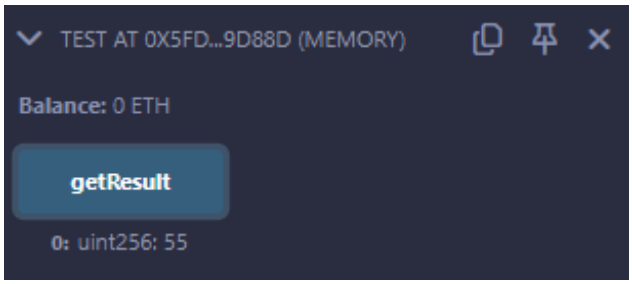
```
pragma solidity ^0.8.26;

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

contract Test is Calculator {
   constructor() public {}

   function getResult() external view returns (uint256) {
      uint256 a = 20;
      uint256 b = 35;
      uint256 result = a + b;
```

```
        return result;
    }
}
```

Output :



Step 1: Click on getResult to display sum



[C] Libraries, Assembly, Events, Error handling.

[I] Libraries

Code :

myLib.sol

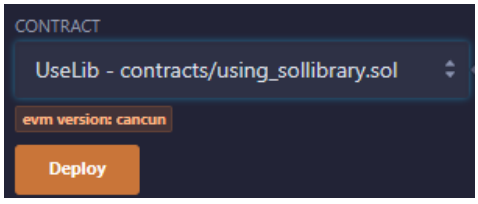
```
pragma solidity 0.8.26;
library myMathLib {
   function sum(uint256 a, uint256 b) public pure returns (uint256) {
      return a + b;
   }
   function exponent(uint256 a, uint256 b) public pure returns (uint256) {
      return a**b;
   }
}
```

using_sollibrary.sol

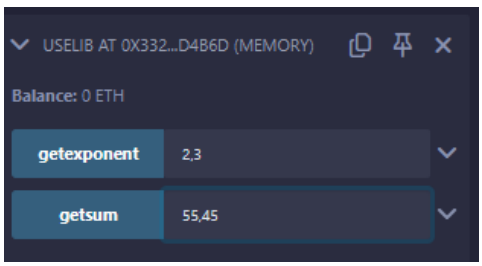
```
pragma solidity >=0.7.0 <0.9.0;
import "contracts/myLib.sol";
contract UseLib {
   function getsum(uint256 x, uint256 y) public pure returns (uint256) {
      return myMathLib.sum(x, y);
   }
   function getexponent(uint256 x, uint256 y) public pure returns (uint256) {
      return myMathLib.exponent(x, y);
   }
}
```

Output :

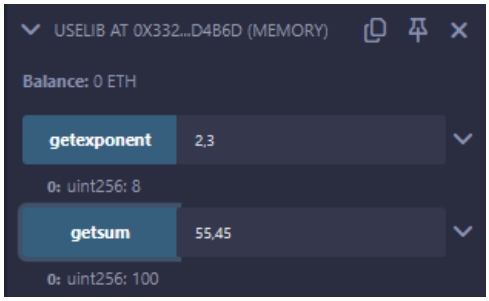
Step 1: Change contract to UseLib and deploy.



Step 2: The deployed contract should be same as below.



Step 3: Add the values, execute both functions. You will get below output.

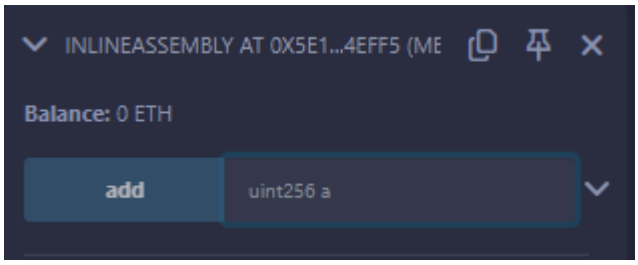


[II] Assembly

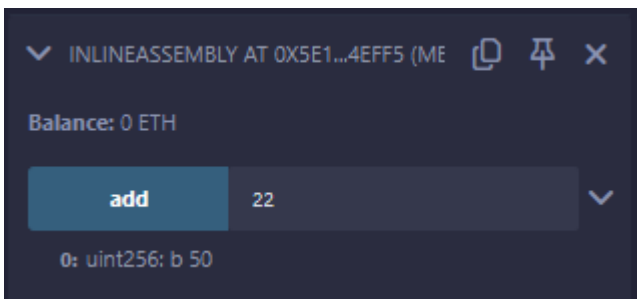
Code :

```
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 :



Step 1: Input a number for add function, click add to output sum

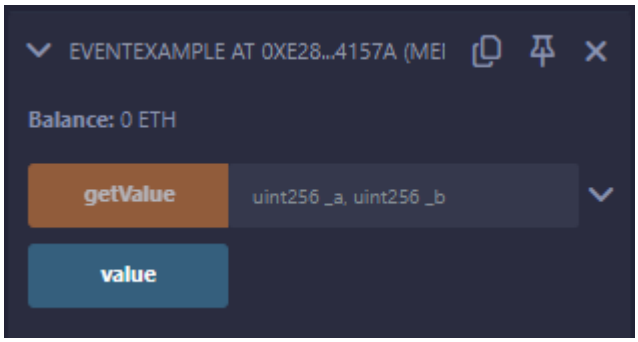


[IV] Events

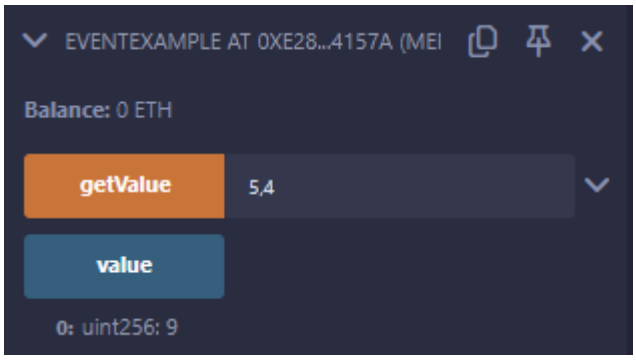
Code :

```
pragma solidity ^0.8.26;
// 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);
      value = _a + _b;
   }
}
```

Output :



Step 1: Provide values to getValue function and click on it.



Step 2: In the terminal check for logs

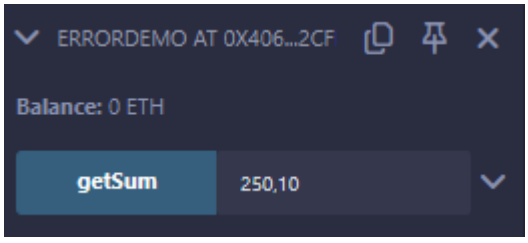
[V] Error handling

Code :

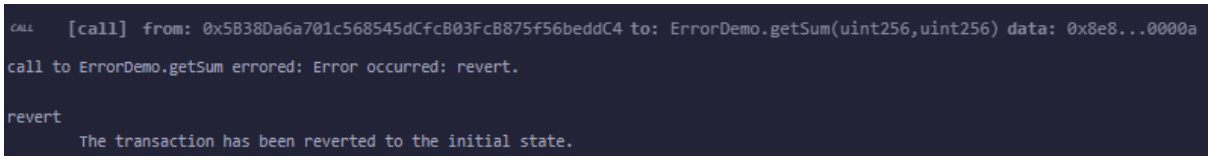
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

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 :

Step 1: Provide some values and press on getSum



Step 2: Check terminal panel



PRACTICAL 5

AIM: Write a program to demonstrate mining of Ether.

Code :

```
from hashlib import sha256

import time

MAX_NONCE = 100000000000

def hashGenerator(text):

    return sha256(text.encode("ascii")).hexdigest()

def mine(block_number, transactions, previous_hash, prefix_zeros):

    prefix_str = '0'*prefix_zeros

    for nonce in range(MAX_NONCE):

        text = str(block_number) + transactions + previous_hash + str(nonce)

        new_hash = hashGenerator(text)

        if new_hash.startswith(prefix_str):

            print(f"Successfully mined Ethers with nonce value : {nonce}")

            return new_hash

    raise BaseException(f"Couldn't find correct hash after trying {MAX_NONCE} times")

if __name__ == '__main__':

    transactions = '''

    Jhon->Paul->77,

    Akon->Bruno->18

    '''

    difficulty = 4

    start = time.time()

    print("Ether mining started.")

    new_hash =
mine(5,transactions,'0000000xa036944e29568d0cff17edbe038f81208fecf9a66be9a2b8321c6
ec7', difficulty)
```

```
    total_time = str((time.time() - start))
    print(f"Ether mining finished.")
    print(f"Ether Mining took : {total_time} seconds")
print(f"Calculated Hash = {new_hash}")
```

Output :

```
======= RESTART: C:\Users\murarilal\Desktop\Blockchain\ethermine\test.py =======
Ether mining started.
Successfully mined Ethers with nonce value : 32474
Ether mining finished.
Ether Mining took : 0.5467884540557861 seconds
Calculated Hash = 00001463e533d7fa091f739026e9fc256c494aff8a1839d41a29448le1f4af82
```

PRACTICAL 6

AIM: Create your own blockchain and demonstrate its use.

Code :

```
from hashlib import sha256

def hashGenerator(text):

   return sha256(text.encode("ascii")).hexdigest()

class Block:

   def __init__(self,data,hash,prev_hash):

      self.data=data

      self.hash=hash

      self.prev_hash=prev_hash

class Blockchain:

   def __init__(self):

     hashLast=hashGenerator('gen_last')

     hashStart=hashGenerator('gen_hash')

     genesis=Block('gen-data',hashStart,hashLast)

     self.chain=[genesis]

   def add_block(self,data):

      prev_hash=self.chain[-1].hash

      hash=hashGenerator(data+prev_hash)

      block=Block(data,hash,prev_hash)

      self.chain.append(block)

bc=Blockchain()

bc.add_block('1')

bc.add_block('2')

bc.add_block('3')
```

for block in bc.chain:

print(block.__dict__)

Output :

```
= RESTART: C:\Users\murarilal\Desktop\Blockchain\ethermine\practical6blockchain.py
{'data': 'gen-data', 'hash': '0a87388e67f16d830a9a3323dad0fdfa4c4044a6a6389cabla0a3
7b651a5717b', 'prev_hash': 'bd6fecc16d509c74d23b04f00f936705e3eaa907b04b78872044607
665018477'}
{'data': '1', 'hash': 'e3e6c97161f3deaf01599fda60ba85593b07f70328bf228473dld408f740
0241', 'prev_hash': '0a87388e67f16d830a9a3323dad0fdfa4c4044a6a6389cabla0a37b651a571
7b'}
{'data': '2', 'hash': '47e8645e3c14bd4034a498aa88ea630bc0793375207bf90ca469792a5d94
84e1', 'prev_hash': 'e3e6c97161f3deaf01599fda60ba85593b07f70328bf228473dld408f74002
41'}
{'data': '3', 'hash': '82084603decbla14a8819dacaa86197659f1e150c4a50186e68043004b5a
3c06', 'prev_hash': '47e8645e3c14bd4034a498aa88ea630bc0793375207bf90ca469792a5d9484
e1'}
```