INDEX FOR BLOCKCHAIN PRACTICAL

Practical No.	Practical	Page No.	Date	Sign
1.	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.			
b.	A transaction class to send and receive money and test it.			
c.	Create multiple transactions and display them.			
d.	Create a blockchain, a genesis block and execute it.			
e.	Create a mining function and test it.			
f.	Add blocks to the miner and dump the blockchain.			
2.	Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.			
3.	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.			
b.	Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions.			

Practical No.	Practical	Page No.	Date	Sign
4	Implement and demonstrate the use of the following in Solidity:			
a.	Withdrawal Pattern, Restricted Access.			
b.	Contracts, Inheritance, Constructors, Abstract Contracts, Interfaces.			
c.	Libraries, Assembly, Events, Error handling.			
5.	Install hyperledger fabric and composer. Deploy and execute the application.			
6.	Write a program to demonstrate mining of Ether.			
7.	Demonstrate the running of the blockchain node.			
8.	Demonstrate the use of Bitcoin Core API.			
9.	Create your own blockchain and demonstrate its use.			
10.	Build Dapps with angular.			

PRACTICAL NO. 1

Aim : Write the following programs for Blockchain in Python:

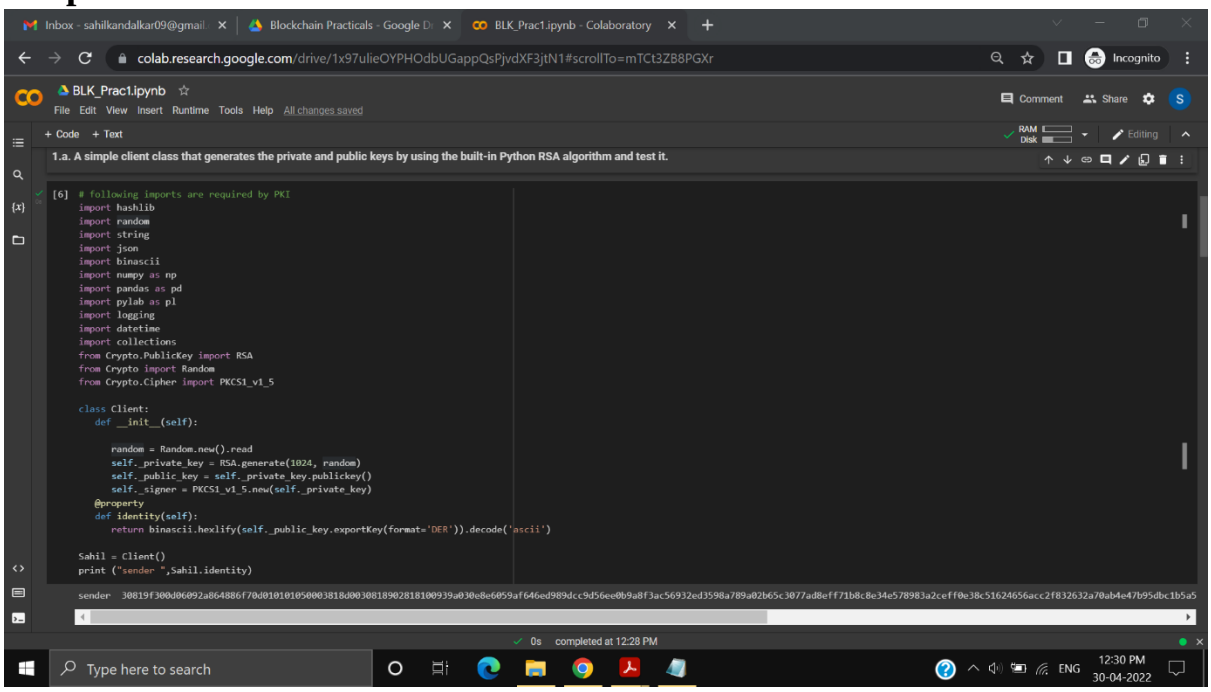
a. A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.
b. A transaction class to send and receive money and test it.
c. Create multiple transactions and display them.
d. Create a blockchain, a genesis block and execute it.
e. Create a mining function and test it.
f. Add blocks to the miner and dump the blockchain.

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

Code :

```
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
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5

class Client:
   def __init__(self):

      random = Random.new().read
      self._private_key = RSA.generate(1024, random)
      self._public_key = self._private_key.publickey()
      self._signer = PKCS1_v1_5.new(self._private_key)
   @property
   def identity(self):
      return binascii.hexlify(self._public_key.exportKey(format='DER'))
.decode('ascii')

Sahil = Client()
print ("sender ",Sahil.identity)
```

Output :



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

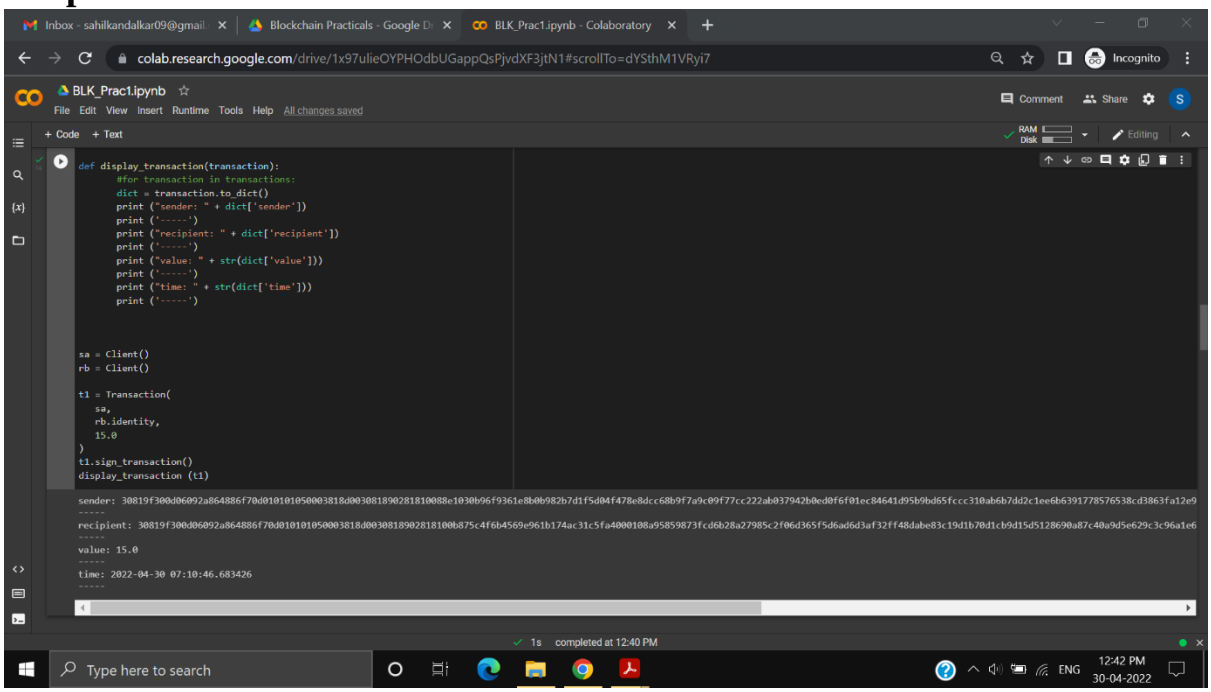
Code :

```
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

from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
from collections import OrderedDict
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5
class Client:
    def __init__(self):
      random = Random.new().read
      self._private_key = RSA.generate(1024, random)
      self._public_key = self._private_key.publickey()
      self._signer = PKCS1_v1_5.new(self._private_key)
```

```
    @property
    def identity(self):
      return binascii.hexlify(self._public_key.exportKey(format='DER'))
.decode('ascii')

class Transaction:
    def __init__(self, sender, 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')) #// hash =MAC
       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 ('-----')


sa = Client()
rb = Client()

t1 = Transaction(
   sa,
```

```
    rb.identity,
    15.0
)
t1.sign_transaction()
display_transaction (t1)
```

Output :



c. Create multiple transactions and display them.

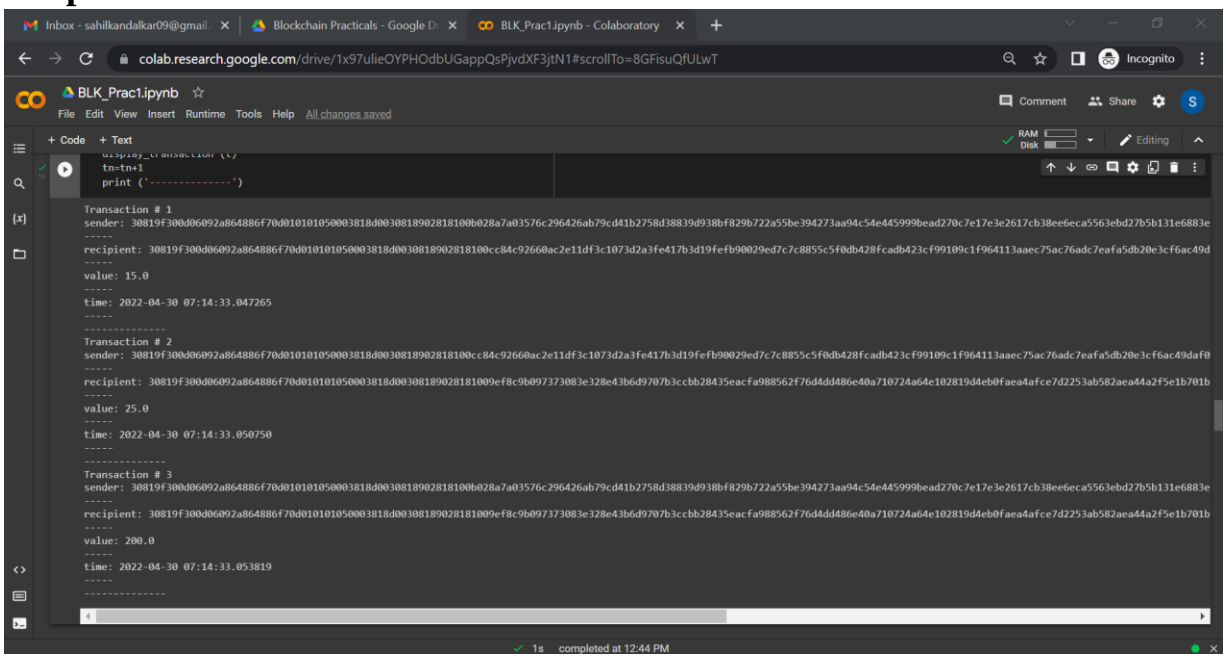
Code :

```
# following imports are required by PKI
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

from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
from collections import OrderedDict
import Crypto
import Crypto.Random
```

```
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5
class Client:
    def __init__(self):
      random = Random.new().read
      self._private_key = RSA.generate(1024, random)
      self._public_key = self._private_key.publickey()
      self._signer = PKCS1_v1_5.new(self._private_key)
    @property
    def identity(self):
      return binascii.hexlify(self._public_key.exportKey(format='DER'))
.decode('ascii')

class Transaction:
    def __init__(self, sender, 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')

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 ('-----')

transactions = []

a = Client()
b = Client()
c = Client()

t1 = Transaction(
   a,
   b.identity,
   15.0
)

t1.sign_transaction()
transactions.append(t1)

t2 = Transaction(
   b,
   c.identity,
   25.0
)
t2.sign_transaction()
transactions.append(t2)

t3 = Transaction(
   a,
   c.identity,
   200.0
)
t3.sign_transaction()
transactions.append(t3)

tn=1
for t in transactions:#t1 t2 t3
   print("Transaction #",tn)
   display_transaction (t)
   tn=tn+1
   print ('--------------')
```

Output :



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

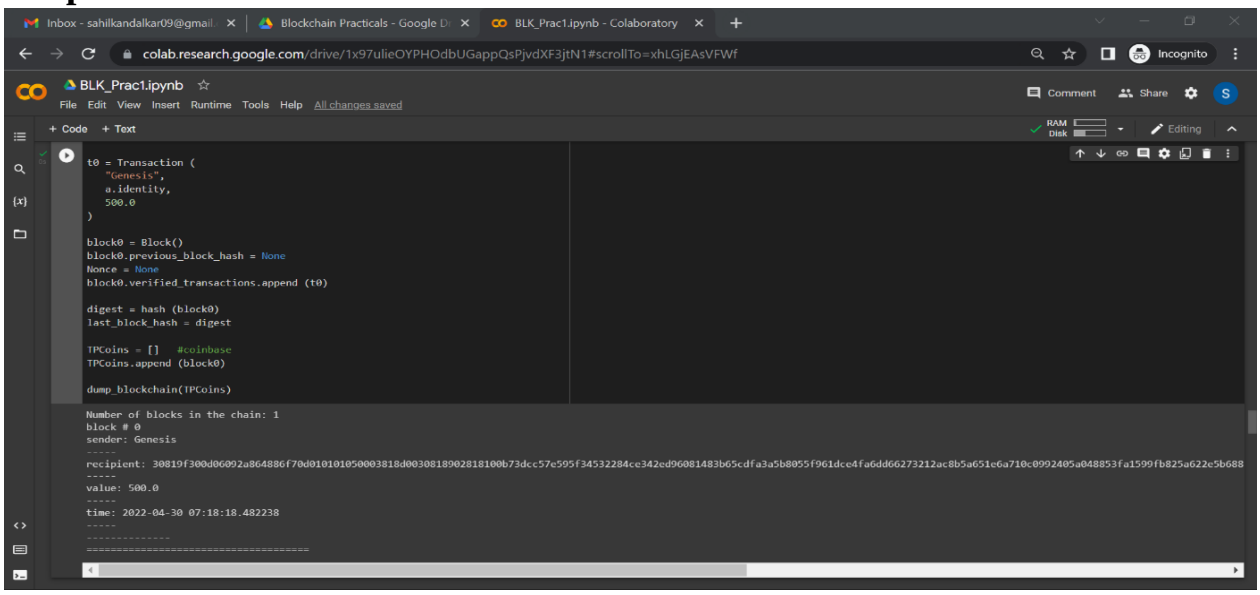
Code :

```
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

from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
from collections import OrderedDict
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5

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

```
    @property
    def identity(self):
      return binascii.hexlify(self._public_key.exportKey(format='DER'))
.decode('ascii')

class Transaction:
    def __init__(self, sender, 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')

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 ('-----')

def dump_blockchain (self):
   print ("Number of blocks in the chain: " + str(len (self)))
   for x in range (len(TPCoins)):
      block_temp = TPCoins[x]
      print ("block # " + str(x))
      for transaction in block_temp.verified_transactions:
```

```
            display_transaction (transaction)
            print ('--------------')
        print ('====================================')

class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""

a = Client()

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

block0 = Block()
block0.previous_block_hash = None
Nonce = None
block0.verified_transactions.append (t0)

digest = hash (block0)
last_block_hash = digest

TPCoins = []    #coinbase
TPCoins.append (block0)

dump_blockchain(TPCoins)
```

Output :

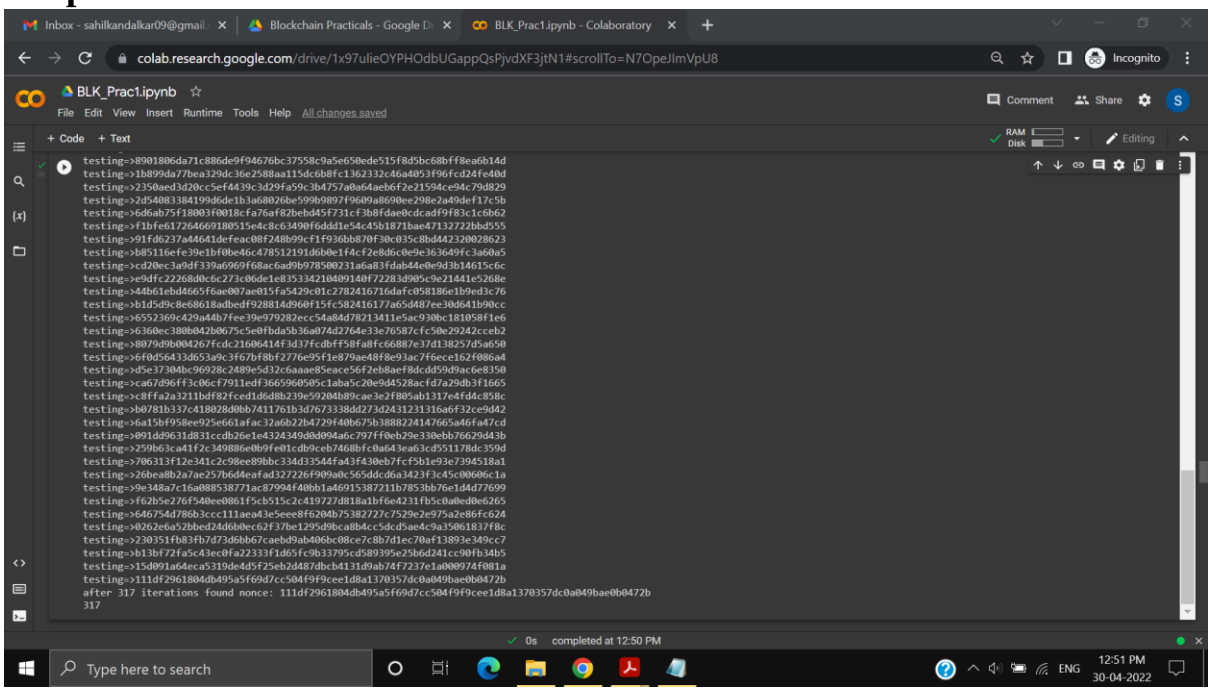


e. Create a mining function and test it.

Code :

```
import hashlib

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

def mine(message, difficulty=1):
   assert difficulty >= 1
   #if(difficulty <1):
   #         return
   #'1'*3=> '111'
   prefix = '1' * difficulty
   print("prefix",prefix)
   for i in range(1000):
      digest = sha256(str(hash(message)) + str(i))
      print("testing=>"+digest)
      if digest.startswith(prefix):
         print ("after " + str(i) + " iterations found nonce: "+ digest
)
         return i #i= nonce value

n=mine ("test message",3)
print(n)
```

Output :



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

Code :

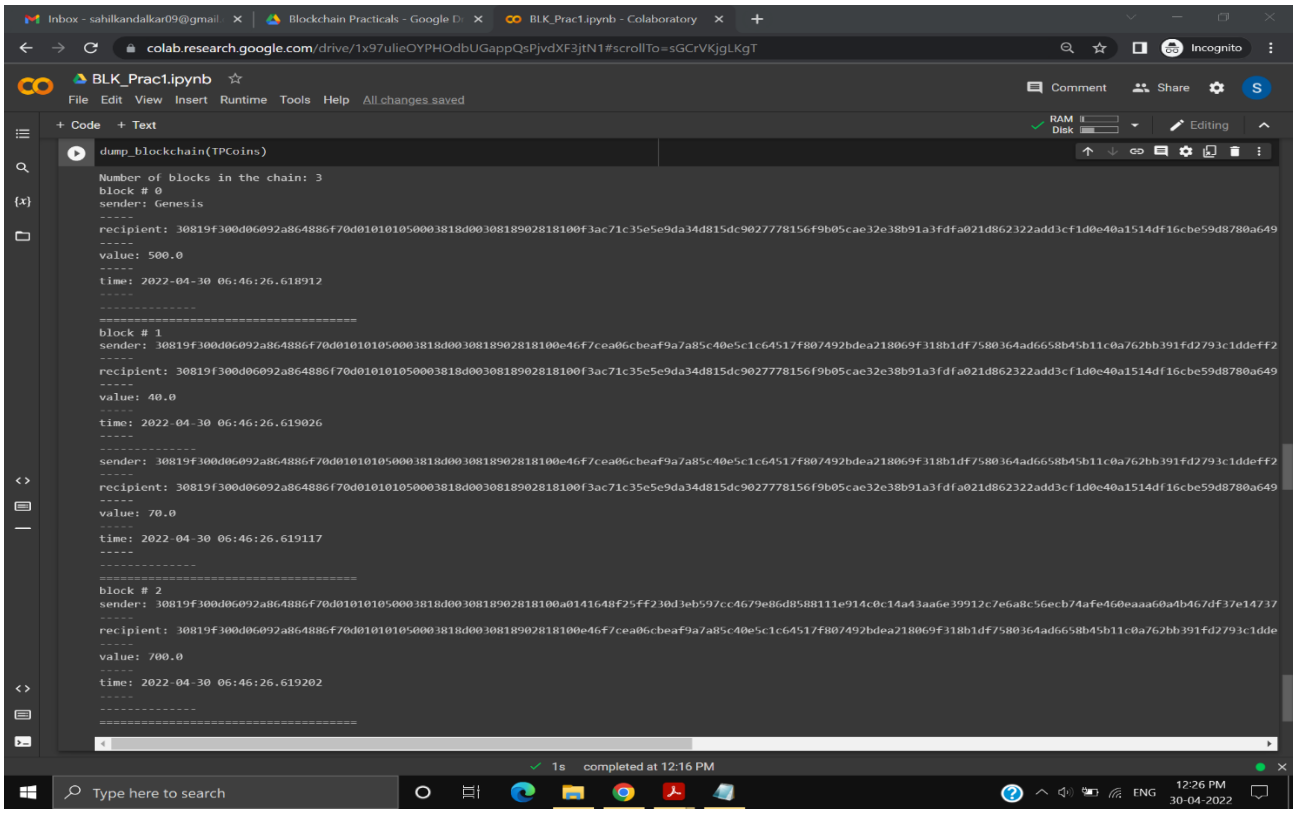
```
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections

from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
from collections import OrderedDict
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5

class Client:
    def __init__(self):
      random = Random.new().read
      self._private_key = RSA.generate(1024, random)
      self._public_key = self._private_key.publickey()
      self._signer = PKCS1_v1_5.new(self._private_key)
    @property
    def identity(self):
      return binascii.hexlify(self._public_key.exportKey(format='DER'))
.decode('ascii')

class Transaction:
    def __init__(self, sender, 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')

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 ('-----')

def dump_blockchain (self):
    print ("Number of blocks in the chain: " + str(len (self)))
    for x in range (len(TPCoins)):
        block_temp = TPCoins[x]
        print ("block # " + str(x))
        for transaction in block_temp.verified_transactions:
            display_transaction (transaction)
            print ('--------------')
        print ('====================================')

class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""

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

def mine(message, difficulty=1):
    assert difficulty >= 1
    #if(difficulty <1):
    #        return
    #'1'*3=> '111'
    prefix = '1' * difficulty
```

```
   for i in range(1000):
      digest = sha256(str(hash(message)) + str(i))
      if digest.startswith(prefix):
         return i #i= nonce value


Sahil = Client()
Rahul =Client()
Aniket =Client()
t0 = Transaction (
   "Genesis",
   Sahil.identity,
   500.0
)

t1 = Transaction (
   Rahul,
   Sahil.identity,
   40.0
)
t2 = Transaction (
   Rahul,
   Sahil.identity,
   70.0
)
t3 = Transaction (
   Aniket,
   Rahul.identity,
   700.0
)
#blockchain
TPCoins = []

block0 = Block()
block0.previous_block_hash = None
Nonce = None
block0.verified_transactions.append (t0)
digest = hash (block0)
last_block_hash = digest #last_block_hash it is hash of block0
TPCoins.append (block0)

block1 = Block()
block1.previous_block_hash = last_block_hash
block1.verified_transactions.append (t1)
block1.verified_transactions.append (t2)
block1.Nonce=mine (block1, 2)
digest = hash (block1)
last_block_hash = digest
TPCoins.append (block1)
```

```
block2 = Block()
block2.previous_block_hash = last_block_hash
block2.verified_transactions.append (t3)
Nonce = mine (block2, 2)
block2.Nonce=mine (block2, 2)
digest = hash (block2)
last_block_hash = digest
TPCoins.append (block2)


dump_blockchain(TPCoins)
```

Output :



PRACTICAL NO. 2

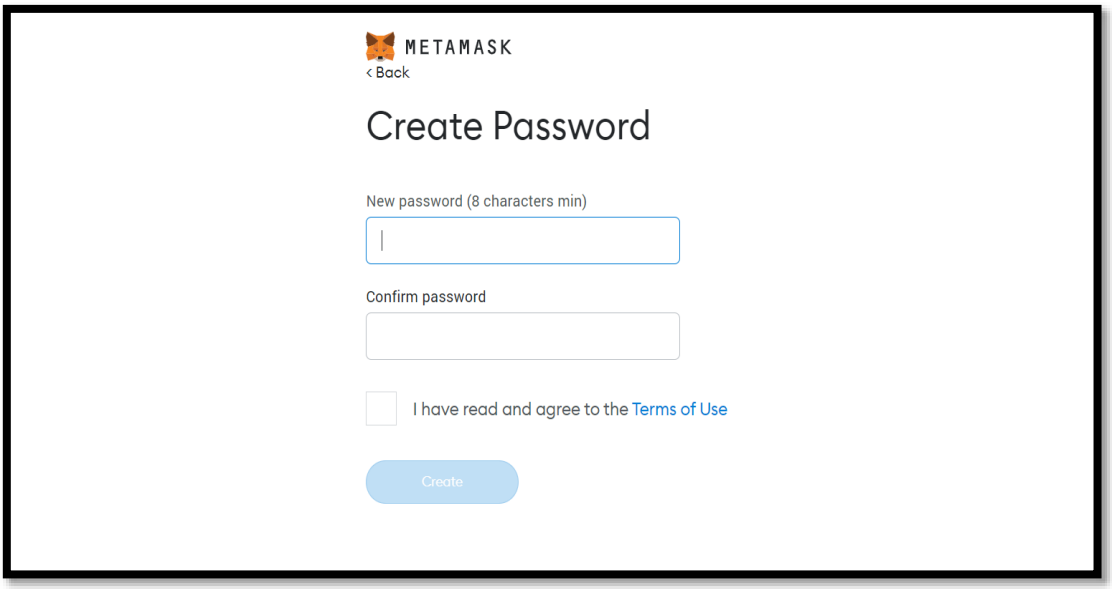
Aim: Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.

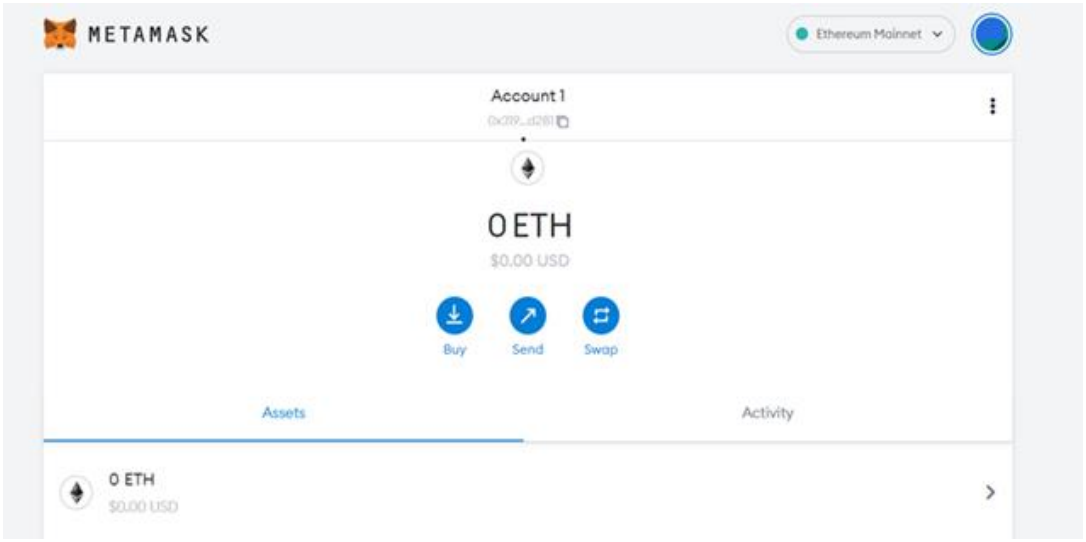
Steps :

A) Ethereum transaction with the help of smart contract in Remix IDE with Rospten test net.

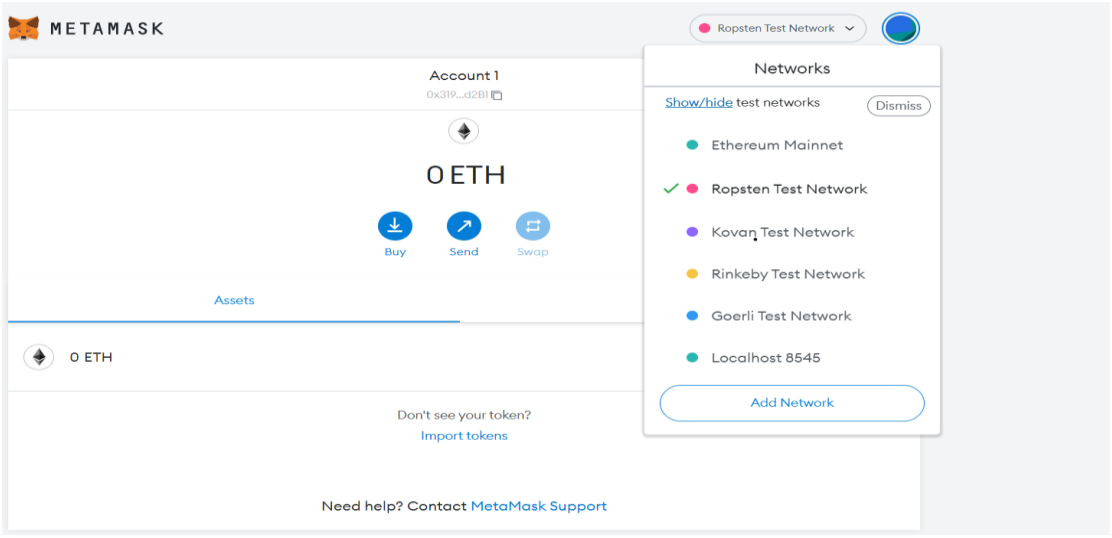
Note: Mist was deprecated in March 2019 after developers decided other browser developers and wallet makers were better able to create products for this quickly evolving space.So in this Remix IDE is used instead of Mist Browser

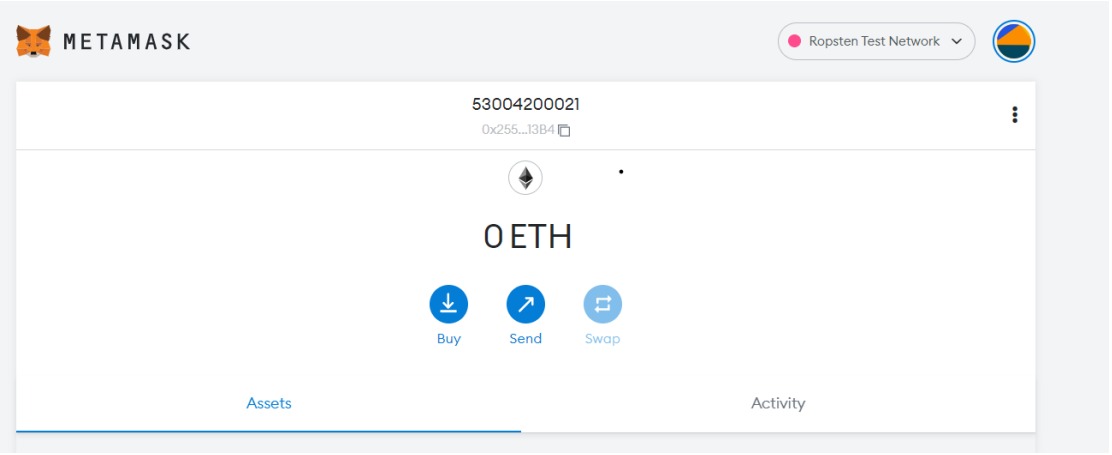
Step 1: Create a account in Meta Mask.

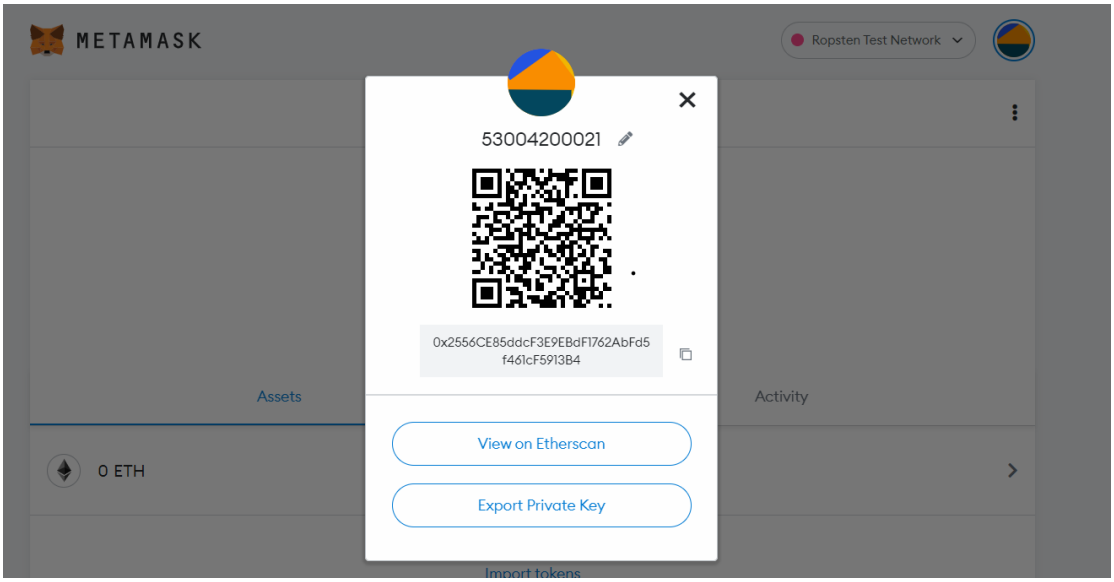




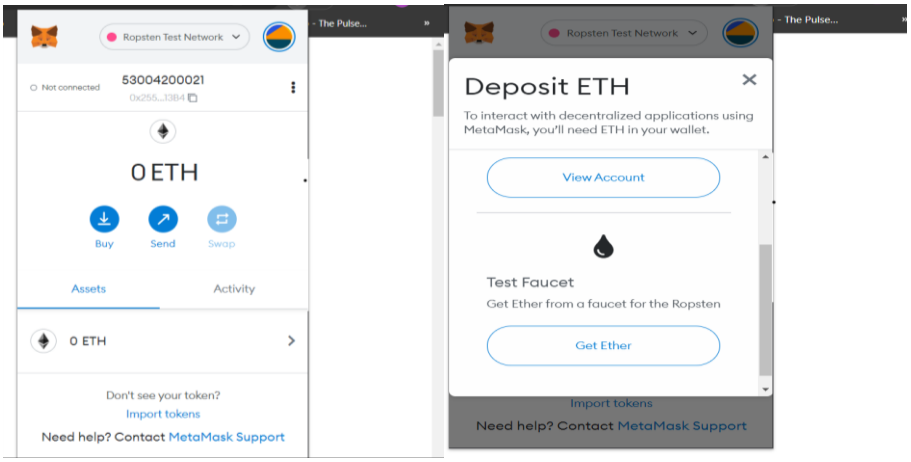
Step 2: To deploy the contract, we need an account and with some ether on the Rospten test network. To open Rospten test network you first need to enable the option to show test networks

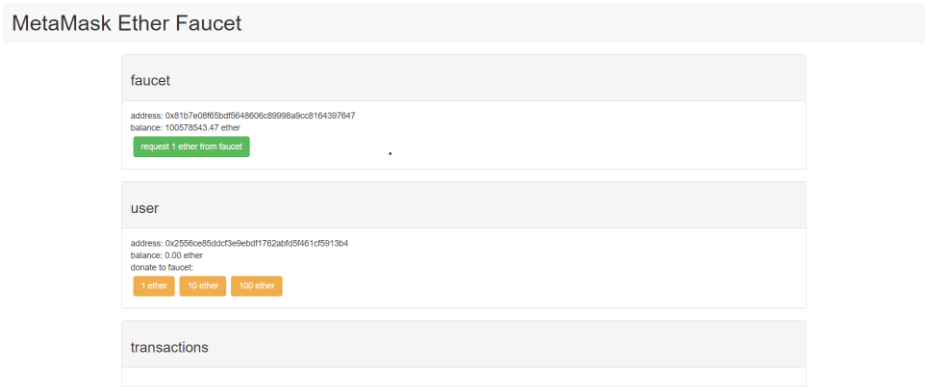


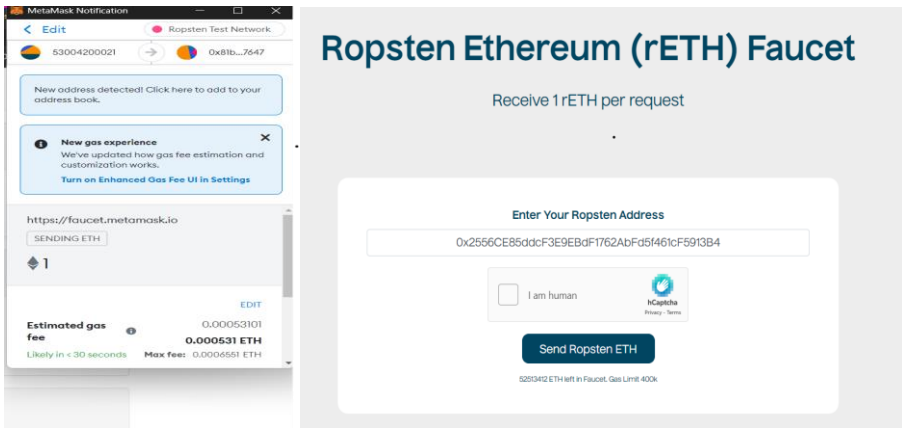




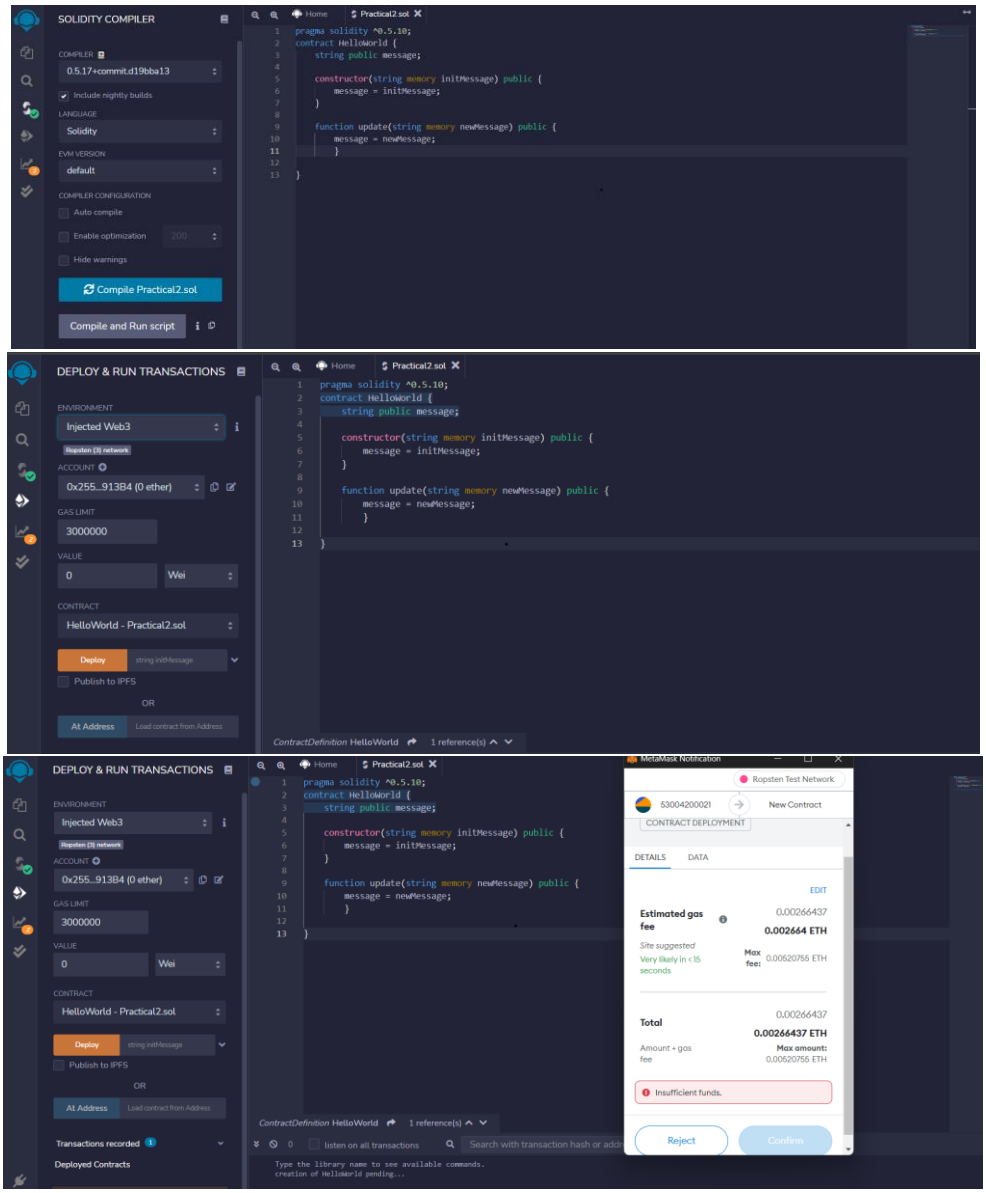
Step 3: Request Free ether from here OR here . Paste the copied account address in text box and click on send me 1 test ether.
Click the MetaMask icon near the top right corner of the browser window.
Select the "Ropsten Test Net" in the dropdown list on the top.
Click "BUY" and "ROPSTEN TEST FAUCET". MetaMask will bring you the https://faucet.metamask.io site with your account address displayed.
Click "Request 1 ether from faucet". You will see the transfer transaction ID displayed at the bottom.







Step 4: Write the code in remix IDE, by creating a new file we have connected meta mask to the ropsten testnet with an account that has some ether.



Code :

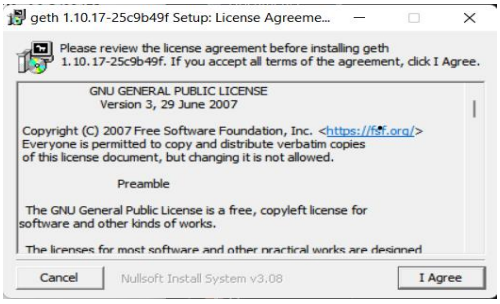
```
pragma solidity ^0.5.10;
contract HelloWorld {
string public message;
constructor(string memory initMessage) public {
message = initMessage;          }
function update(string memory newMessage) public {
message = newMessage;          }
}
```

Step 5: Click on the contract deployment — it should open up the etherscan page to look at our transaction details.

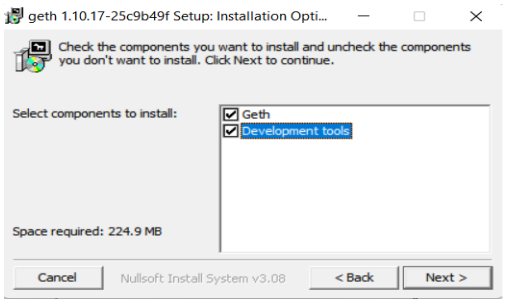
B) Install and Configure Geth and creating a private blockchain and connecting with Remix.

Step 6: Download geth node from https://geth.ethereum.org/downloads/ double click on the .exe file.Click on Agree.

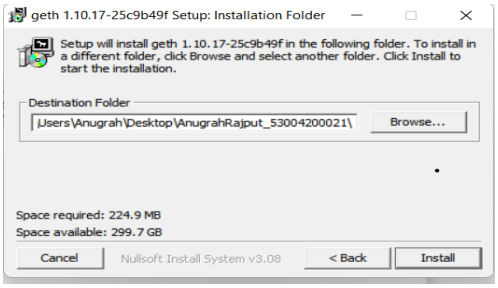
Step 7: Click on I Agree



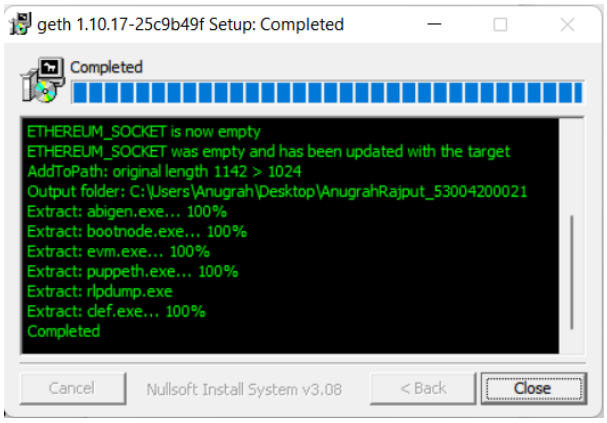
Step 8: Click on Next



Step 9: Specify the path and click on install.

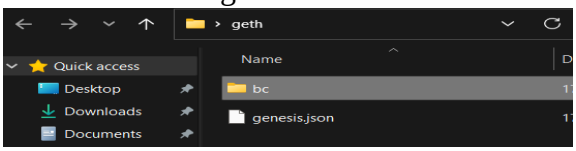


Step 10: Once installed open command prompt write geth version to see if installation is done properly



Step 11: Create a geth folder in that Open a notepad and write the code to create a genesis block and save it as .json file

Step 12: Run this command to create the block chain :
geth --datadir C:\Users\Anugrah\Desktop\geth\bc init C:\Users\Anugrah\Desktop\geth\genesis.json
Now we will find a blockchain file in the geth folder.

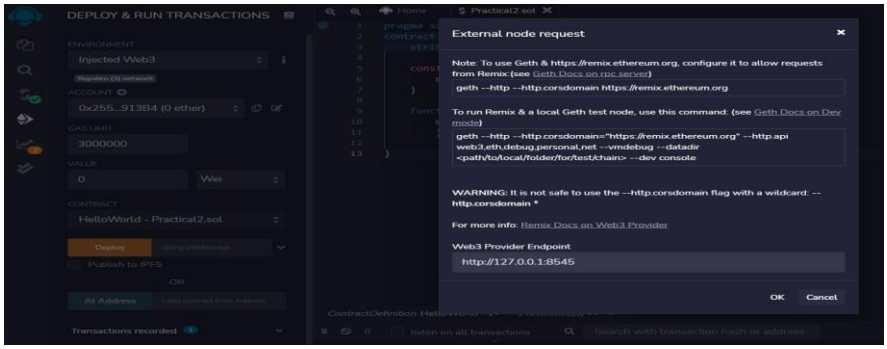


Step 13: Now run this command to activate the blockchain
geth --identity "localB" --rpc --rpcport "8280" --rpccorsdomain "*" -- rpcapi "db,eth,net,web3" --datadir "C:\Users\Anugrah\Desktop\geth\bc" - -port "30303" --nodiscover --networkid 1999 console

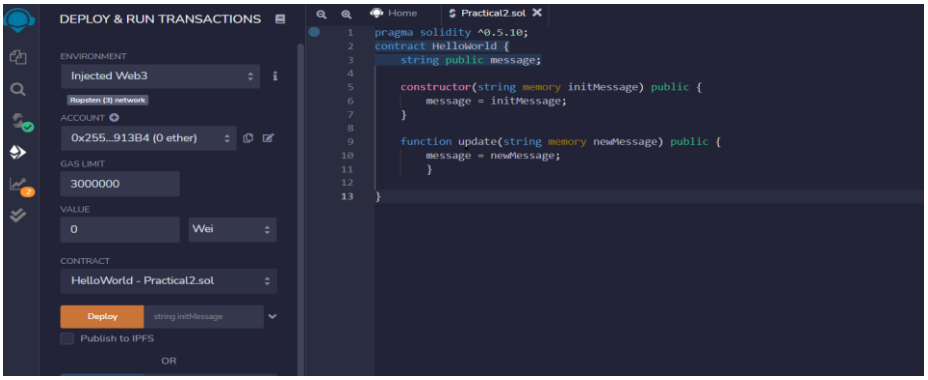
Step 14: Create a account and check the balance of the using the following code :
personal.newAccount()
eth.getBalance("the account number generated while creating the account")

Step 15: Now start the mining using miner.start() command
Now we can connect this blockchain to remix IDE with the help of the port that we have specified in the gensis file

Step 16: Click on deploy and run transactions in Remix IDE click on Injected Web 3 Specify the port number



Now you can see the account number has changed and now you can easily deploy the smart contracts and do transactions.



PRACTICAL NO. 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. Variables: Solidity supports three types of variables.

- State Variables − Variables whose values are permanently stored in a contract storage.
- Local Variables − Variables whose values are present till function is executing.
- Global Variables − Special variables exists in the global namespace used to get information about the blockchain.

i.e. blockhash(uint blockNumber) returns (bytes32), block.coinbase (address payable), block.difficulty (uint)…..and many more

Scope of local variables is limited to function in which they are defined but State variables can have three types of scopes:

- Public − Public state variables can be accessed internally as well as via messages. For a public state variable, an automatic getter function is generated.
- Internal − Internal state variables can be accessed only internally from the current contract or contract deriving from it without using this.
- Private − Private state variables can be accessed only internally from the current contract they are defined not in the derived contract from it.

Code :

1] State 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 = 6;
uint result = a + b;
return result; //access the state variable        }}
```

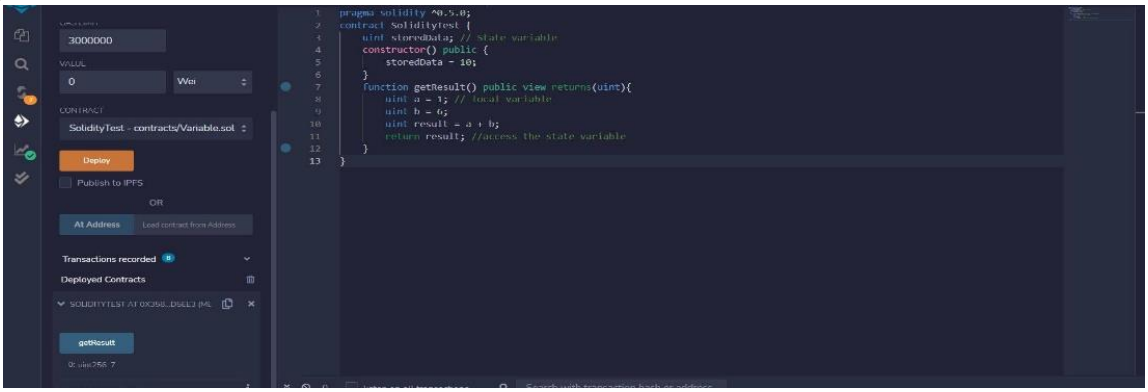
2] Local Variable

```
// Solidity program to demonstrate local variables pragma solidity ^0.5.0;
function getResult() public view returns(uint){
uint local_var1 = 1;
uint local_var2 = 2;
```

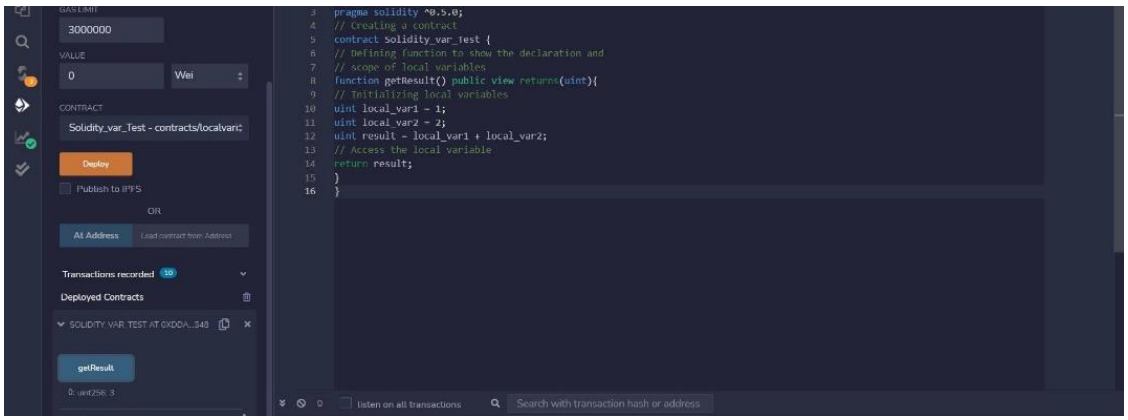
uint result = local_var1 + local_var2; // Access the local variable
return result; }}
3] Global Variable
// Solidity program to show Global variables
pragma solidity ^0.5.0;
contract Test {
address public admin;
constructor() public { admin = msg.sender; }}

Output :

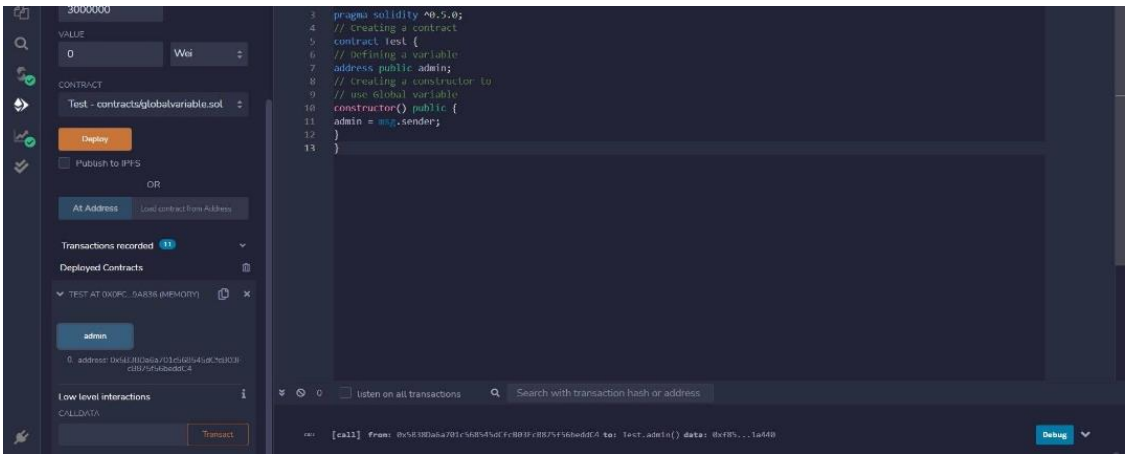
1] State Variable



2] Local Variable



3] Global Variable



II. Operators:

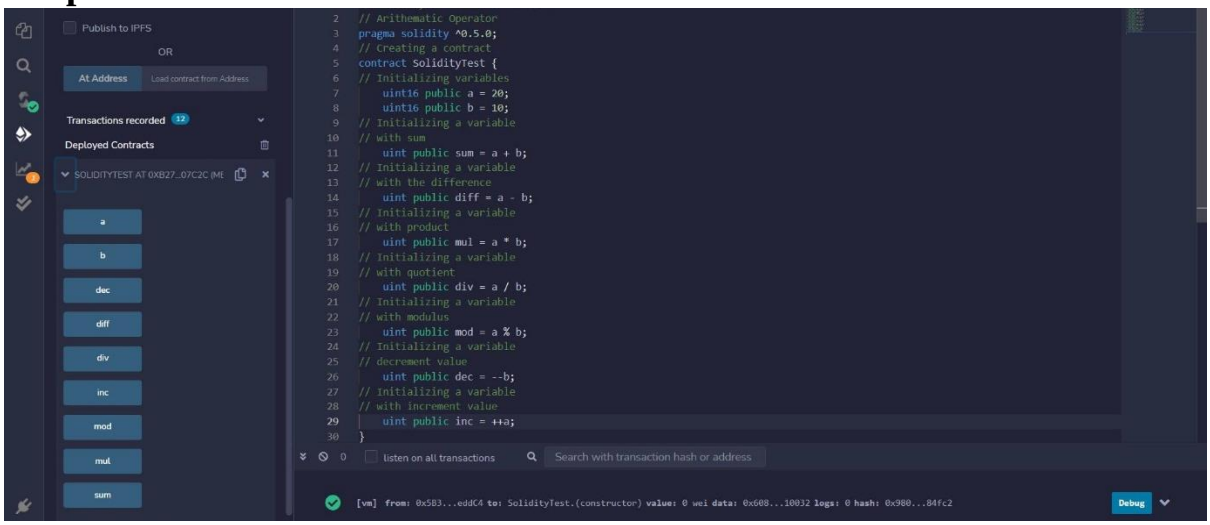
Solidity supports the following types of operators - Arithmetic Operators, Comparison Operators, Logical (or Relational) Operators, Assignment Operators, Conditional (or ternary) Operators.

Arithmetic Operator

Code :

```
// Solidity contract to demonstrate Arithematic Operator
pragma solidity ^0.5.0;
contract SolidityTest {
uint16 public a = 20; uint16 public b = 10;
uint public sum = a + b;
uint public diff = a - b;
uint public mul = a * b;
uint public div = a / b;
uint public mod = a % b;
uint public dec = --b;
value uint public inc = ++a;
}
```

Output :



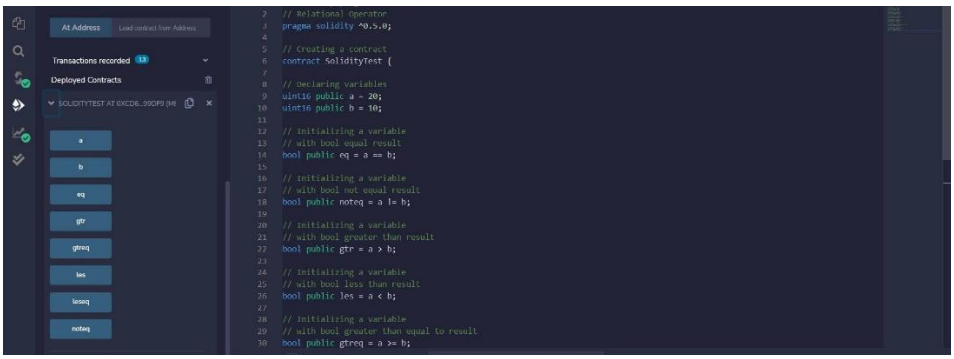
Relational Operator

Code :

```
// Solidity program to demonstrate Relational Operator
pragma solidity ^0.5.0;
contract SolidityTest {
uint16 public a = 20; uint16 public b = 10;
bool public eq = a == b;
bool public noteq = a != b;
```

```
bool public gtr = a > b;
bool public les = a < b;
bool public gtreq = a >= b;
bool public leseq = a <= b;
}
```

Output :

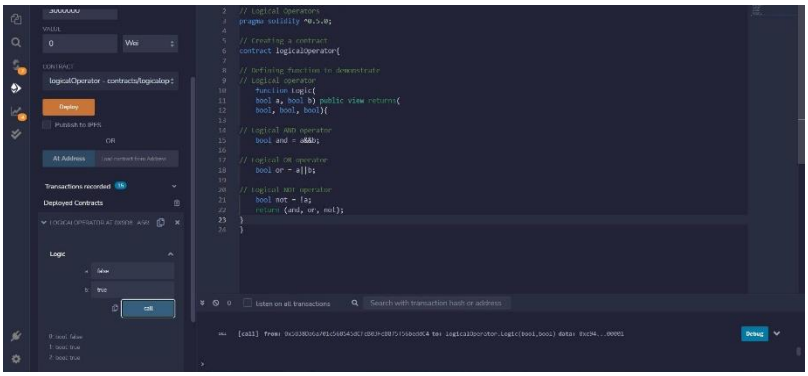


Logical Operator

Code :

```
// Solidity program to demonstrate Logical Operators
pragma solidity ^0.5.0;
contract logicalOperator{
function Logic(bool a, bool b) public view returns( bool, bool, bool){
bool and = a&&b;
bool or = a||b;
bool not = !a;
return (and, or, not);        }}
```

Output :



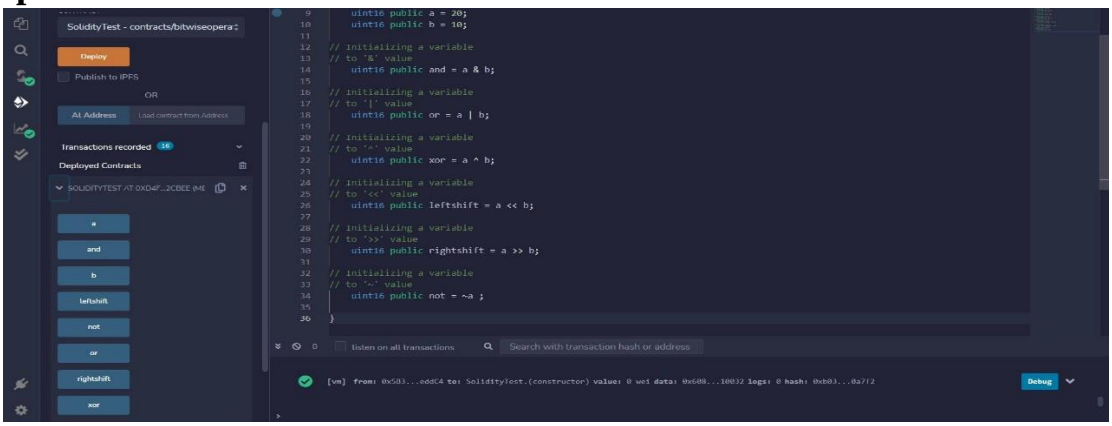
Bitwise Operator

Code :

```
// Solidity program to demonstrate Bitwise Operator
pragma solidity ^0.5.0;
contract SolidityTest {
```

```
uint16 public a = 20; uint16 public b = 10;
uint16 public and = a & b;
uint16 public or = a | b;
uint16 public xor = a ^ b;
// Initializing a variable
// to '<<' value
uint16 public leftshift = a << b;
// Initializing a variable
// to '>>' value
uint16 public rightshift = a >> b;
// Initializing a variable
// to '~' value

uint16 public not = ~a ;
}
```

Output :

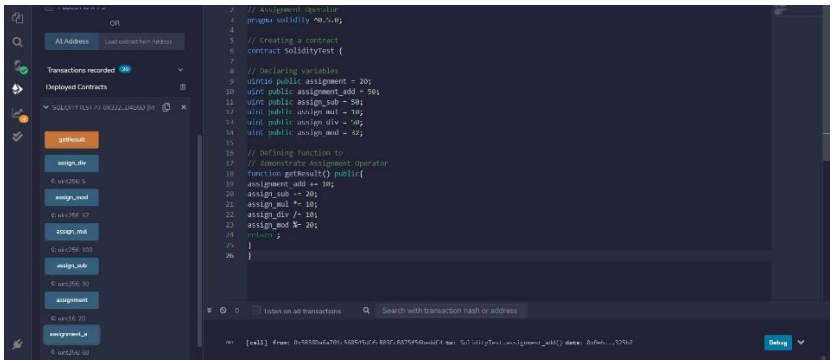


Assignment Operator

Code :

```
// Solidity program to demonstrate Assignment Operator
pragma solidity ^0.5.0;
contract SolidityTest {
uint16 public assignment = 20; uint public assignment_add = 50; uint public
assign_sub = 50;
uint public assign_mul = 10; uint public assign_div = 50; uint public
assign_mod = 32;
function getResult() public{ assignment_add += 10;
assign_sub -= 20;
assign_mul *= 10;
assign_div /= 10;
assign_mod %= 20; return ;       }       }
```

Output :

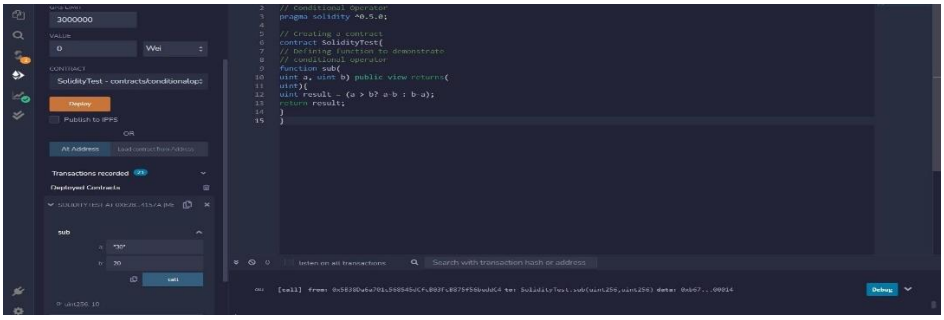


Conditional Operator

Code :

```
// Solidity program to demonstrate Conditional Operator
pragma solidity ^0.5.0;
contract SolidityTest{
function sub(
uint a, uint b) public view returns( uint){
uint result = (a > b? a-b : b-a); return result;  }}
```

Output :



III. Loops:

- While loop: The most basic loop in Solidity is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.
- Do-while loop: The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.
- For loop: The for loop is the most compact form of looping. It includes the following three important parts:

-The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

-The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
-The iteration statement where you can increase or decrease your counter.

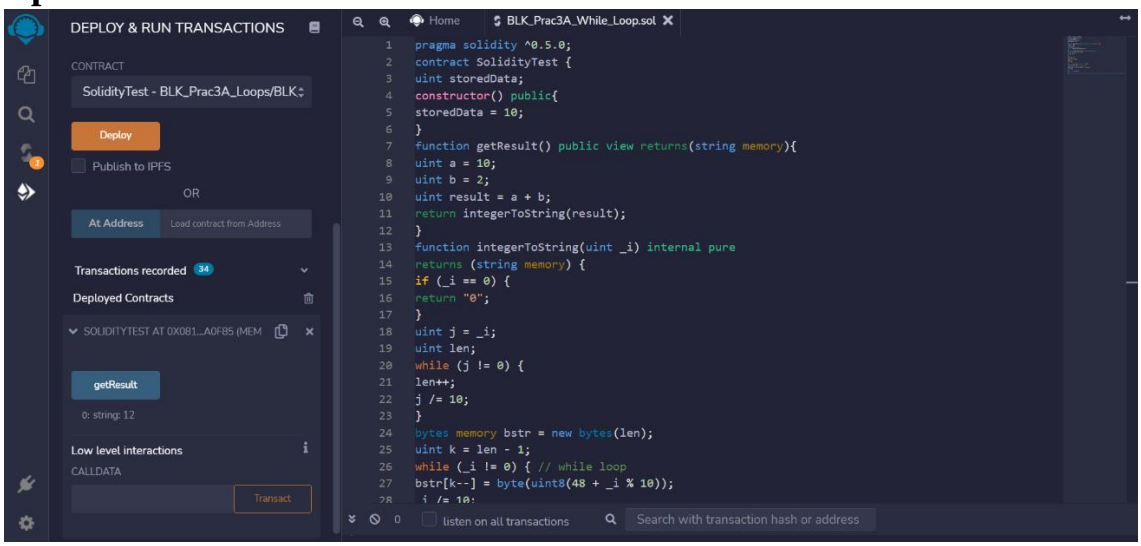
Loop control: Solidity provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.To handle all such situations, Solidity provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

While Loop

Code :

```
pragma solidity ^0.5.0;
contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}
function getResult() public view returns(string memory){
uint a = 10;
uint b = 2;
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) { // while loop
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
}
return string(bstr);     }}
```

Output :

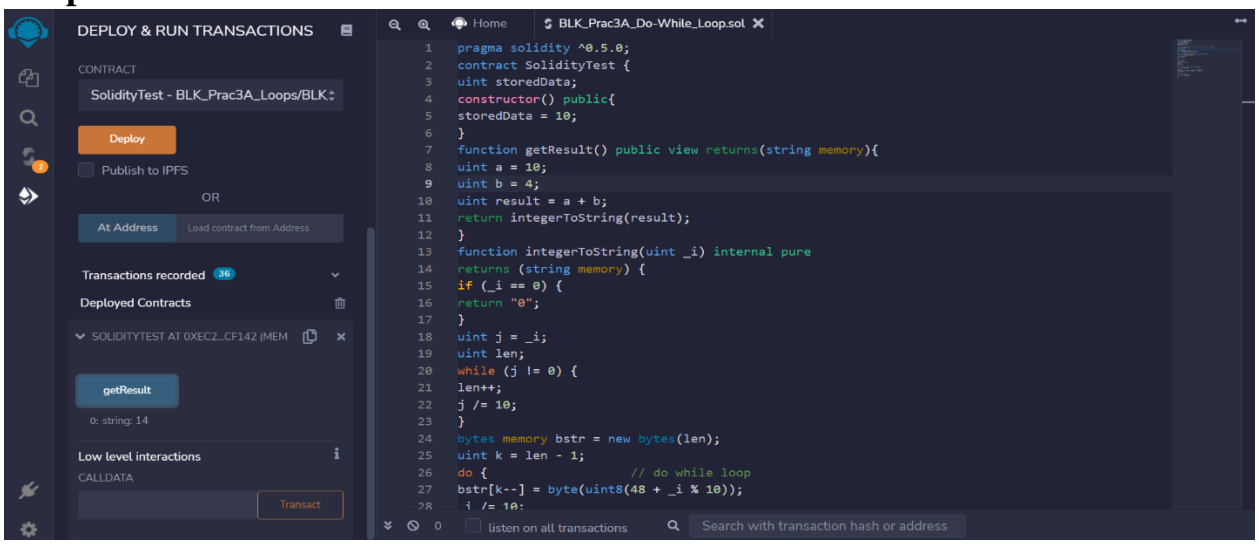


Do-While Loop

Code :

```
pragma solidity ^0.5.0;
contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;   }

function getResult() public view returns(string memory){
uint a = 10; uint b = 4;
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;

do {                    // do while loop
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;    }
while (_i != 0);
return string(bstr);     }}
```

Output :

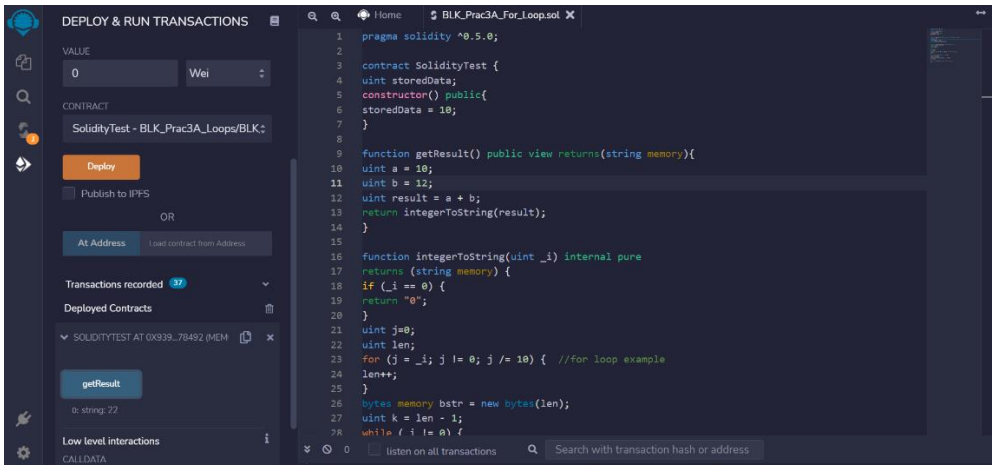


For Loop

Code :

```
pragma solidity ^0.5.0;

contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;   }

function getResult() public view returns(string memory){
uint a = 10;
uint b = 12;
uint result = a + b;
return integerToString(result);       }

function integerToString(uint _i) internal pure
returns (string memory) {
if (_i == 0) {
return "0";  }
uint j=0;
uint len;
for (j = _i; j != 0; j /= 10) {  //for loop example
len++;
}
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);//access local variable       }}
```

Output :



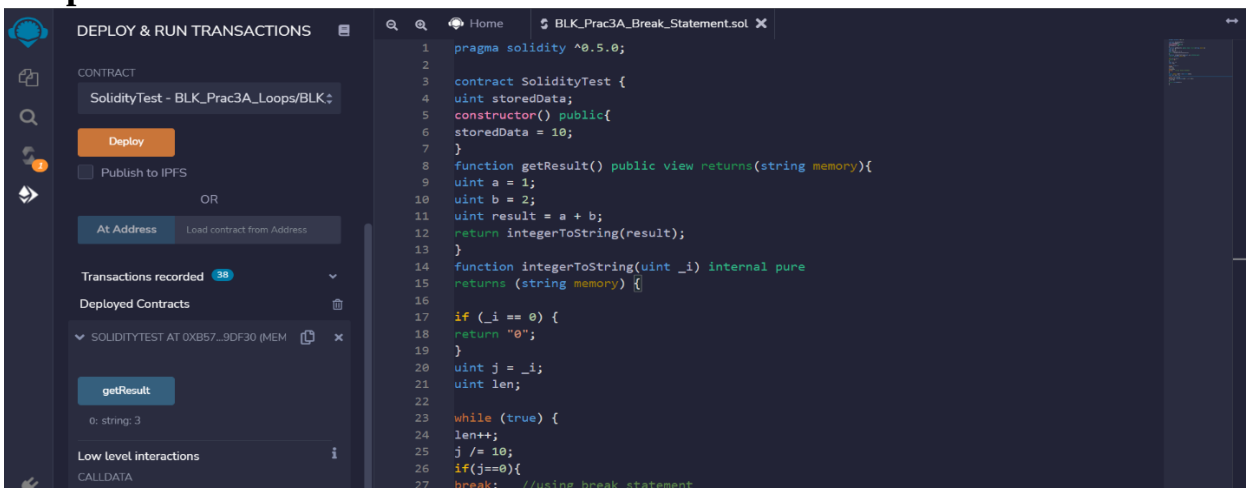
Loop control : break

Code :

```
pragma solidity ^0.5.0;

contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}
function getResult() public view returns(string memory){
uint a = 1;
uint b = 2;
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 (true) {
len++;
j /= 10;
if(j==0){
break;   //using break statement
}
}
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 :



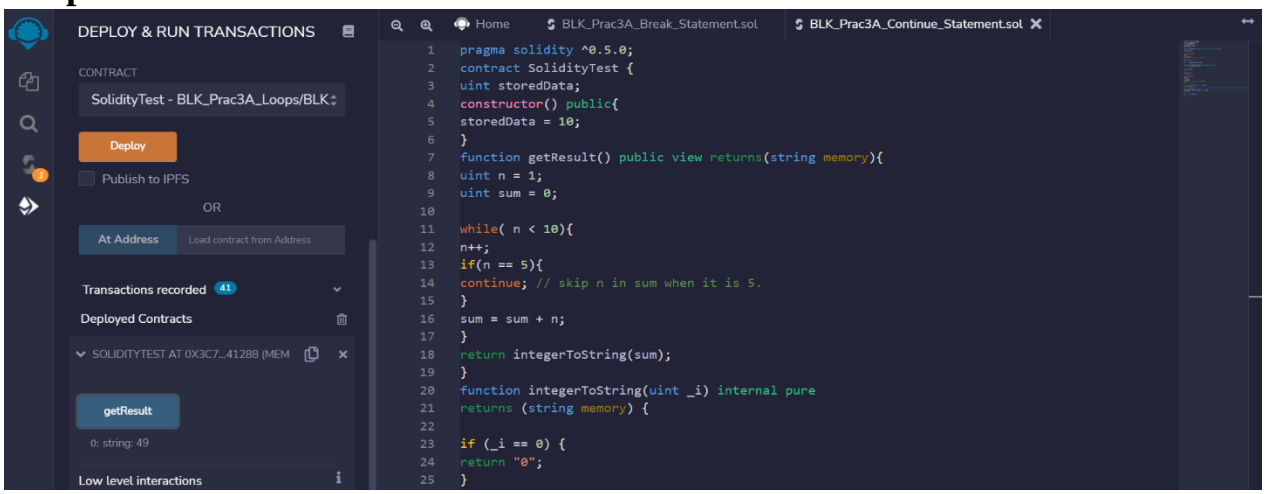
Loop control : continue

Code :

```
contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}
function getResult() public view returns(string memory){
uint n = 1;
uint sum = 0;

while( n < 10){
n++;
if(n == 5){
continue; // skip n in sum when it is 5.
}
sum = sum + n;
}
return integerToString(sum);
}
function integerToString(uint _i) internal pure
returns (string memory) {
```

```
if (_i == 0) {
return "0";
}
uint j = _i;
uint len;

while (true) {
len++;
j /= 10;
if(j==0){
break;   //using break statement
}
}
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 :



IV. Decision Making:

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions. Solidity supports conditional statements which are used to perform different actions based on different conditions.

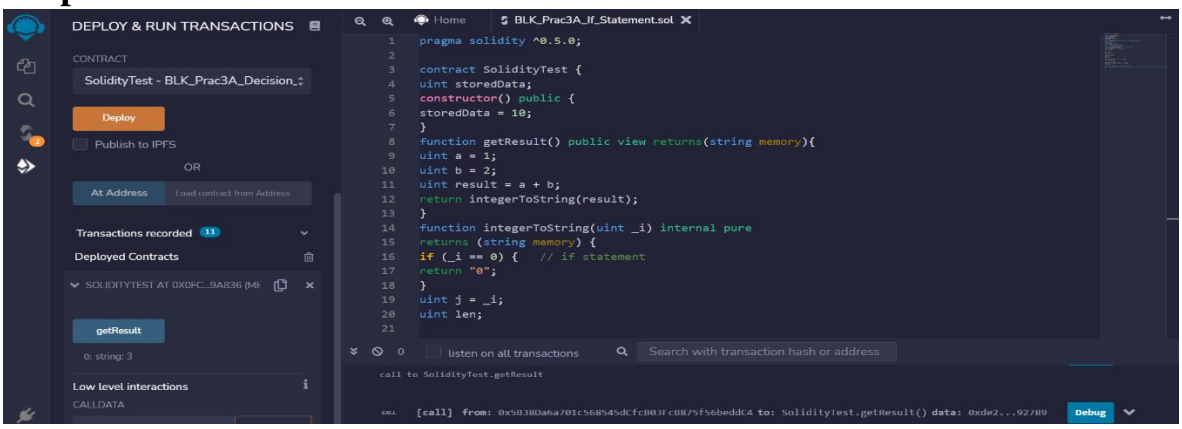
- if statement: The if statement is the fundamental control statement that allows Solidity to make decisions and execute statements conditionally.
- if-else statement: The 'if...else' statement is the next form of control statement that allows Solidity to execute statements in a more controlled way.
- if-else..if statement: The if...else if... statement is an advanced form of if...else that allows Solidity to make a correct decision out of several conditions.

if statement

Code:

```
pragma solidity ^0.5.0;

contract SolidityTest {
uint storedData;
constructor() public {
storedData = 10;   }
function getResult() public view returns(string memory){
uint a = 1; uint b = 2;
uint result = a + b;
return integerToString(result);       }
function integerToString(uint _i) internal pure
returns (string memory) {
if (_i == 0) {   // if statement
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);//access local variable       }}
```

Output :

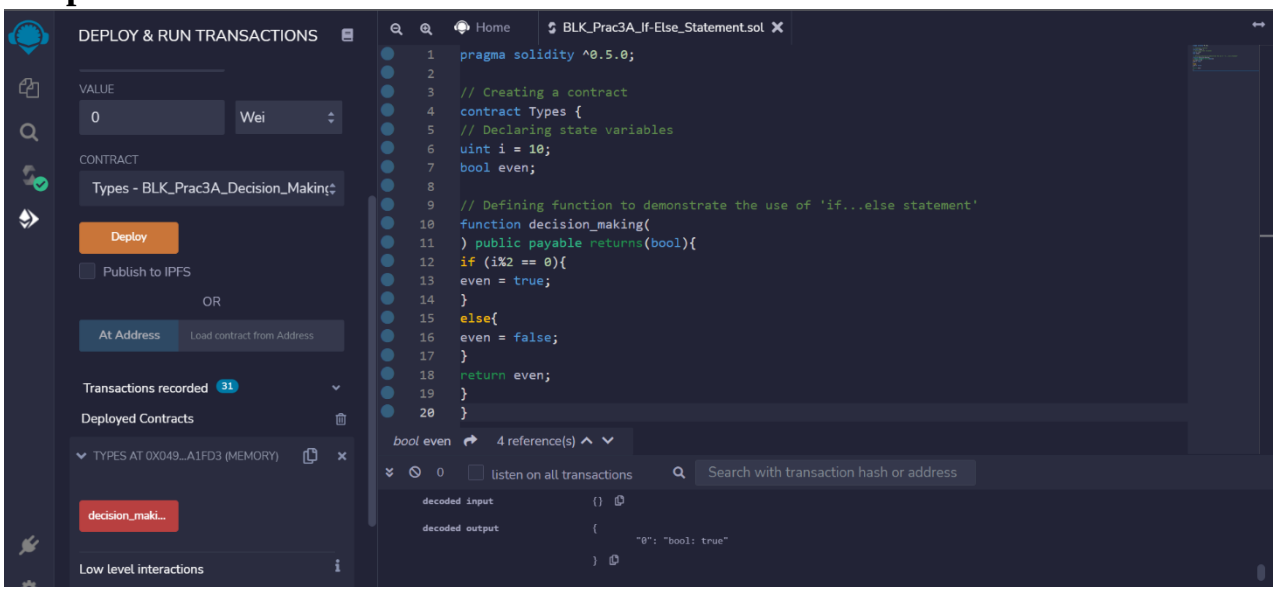


if-else statement

Code:

```
pragma solidity ^0.5.0;

// Creating a contract
contract Types {
// Declaring state variables
uint i = 7; bool even;

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

Output :

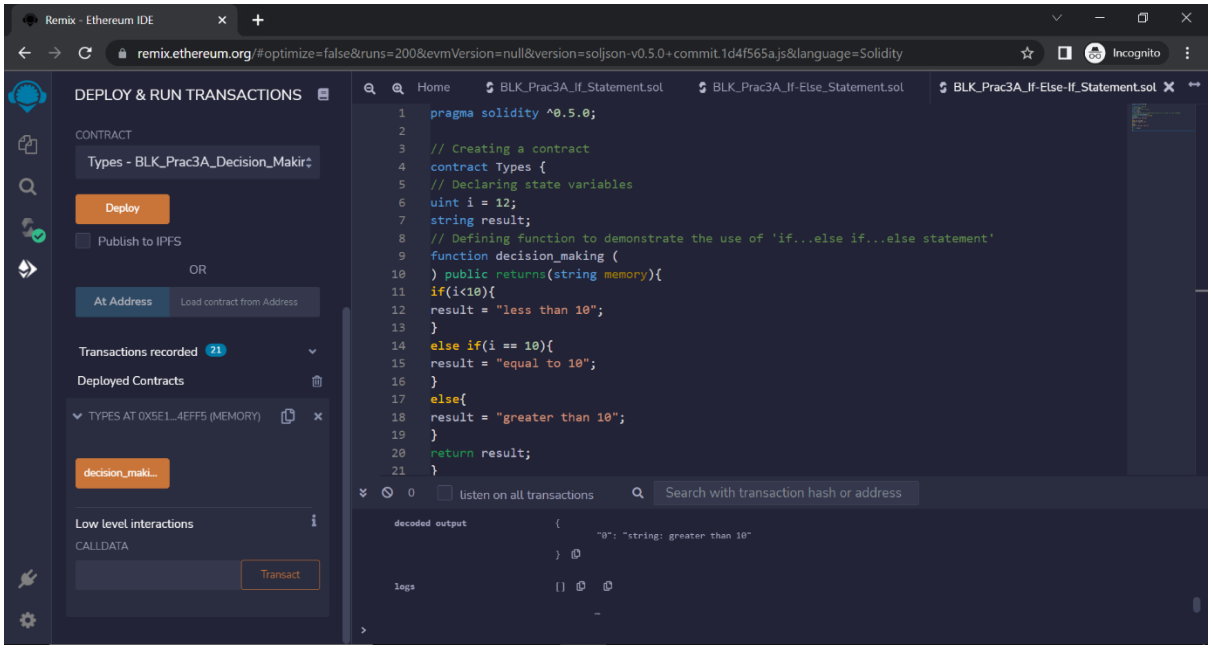


if-else-if statement

Code:

```
pragma solidity ^0.5.0;

// Creating a contract
contract Types {
// Declaring state variables
uint i = 12;
string result;
// Defining function to demonstrate the use of 'if...else if...else statement'
function decision_making () public 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 String:

Solidity supports String literal using both double quote (") and single quote (').
It provides string as a data type to declare a variable of type String.(Int to str)

Code:

```
pragma solidity ^0.5.0; contract SolidityTest {
constructor() public{}
function getResult() public view returns(string memory){
uint a = 1; uint b = 2; 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 :

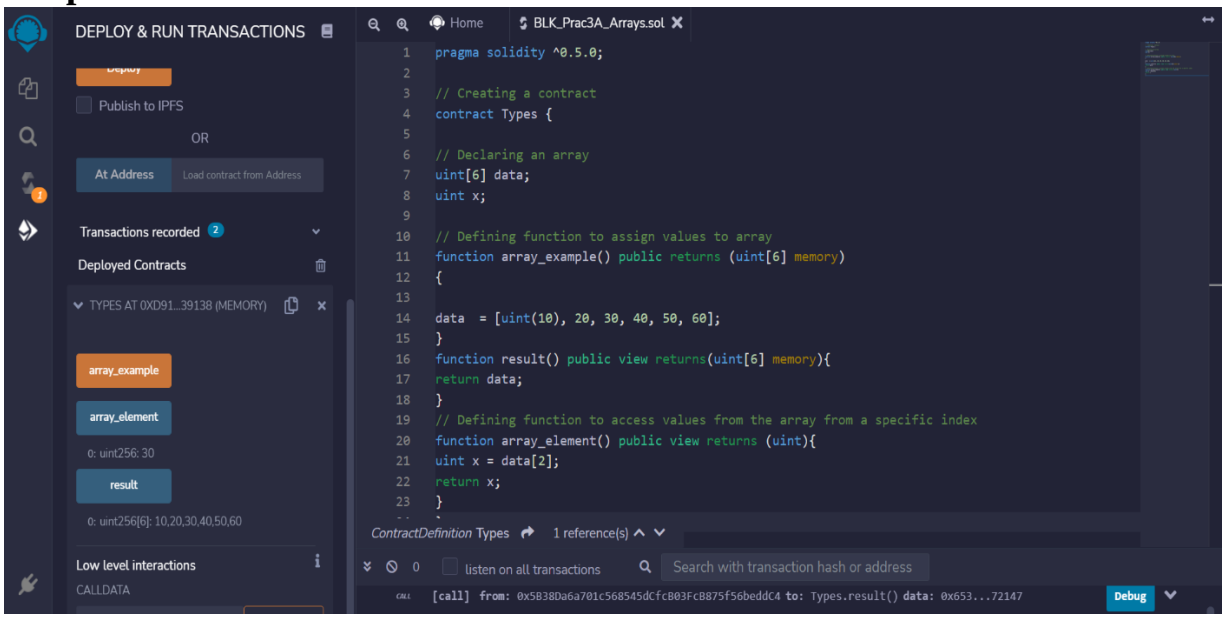
Array:

Array is a data structure, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Code:

```
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

// Declaring an array
uint[6] data; uint x;

// Defining function to assign values to array
function array_example() public returns (uint[6] memory)
{      data  = [uint(10), 20, 30, 40, 50, 60];     }
function result() public view returns(uint[6] memory){
return data;       }
// Defining function to access values from the array from a specific index
function array_element() public view returns (uint){
uint x = data[2];
return x;    }}
```

Output :



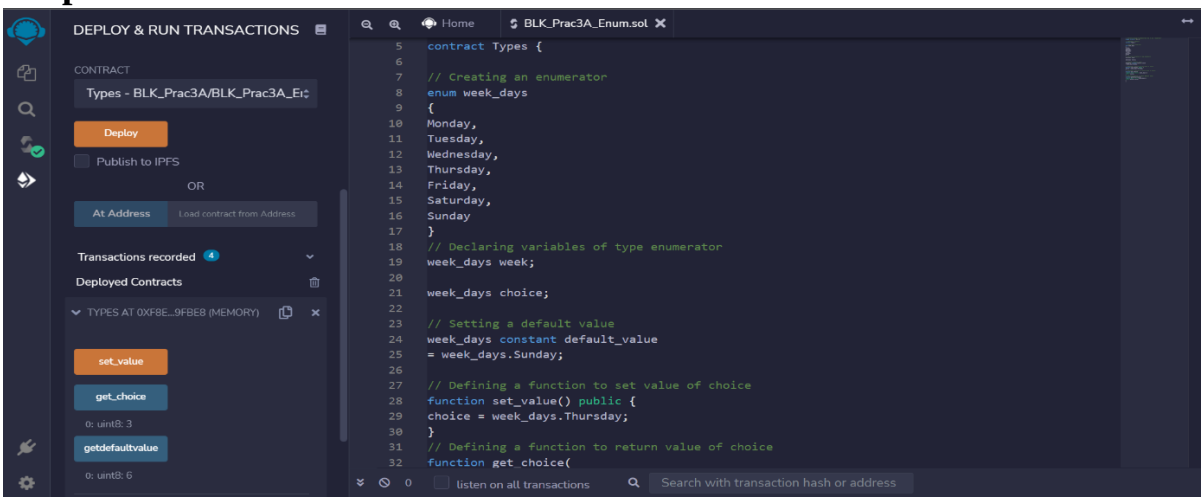
Enums:

Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums. With the use of enums it is possible to reduce the number of bugs in your code.

Code:

```
// Solidity program to demonstrate how to use 'enumerator'
pragma solidity ^0.5.0;
// 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 :

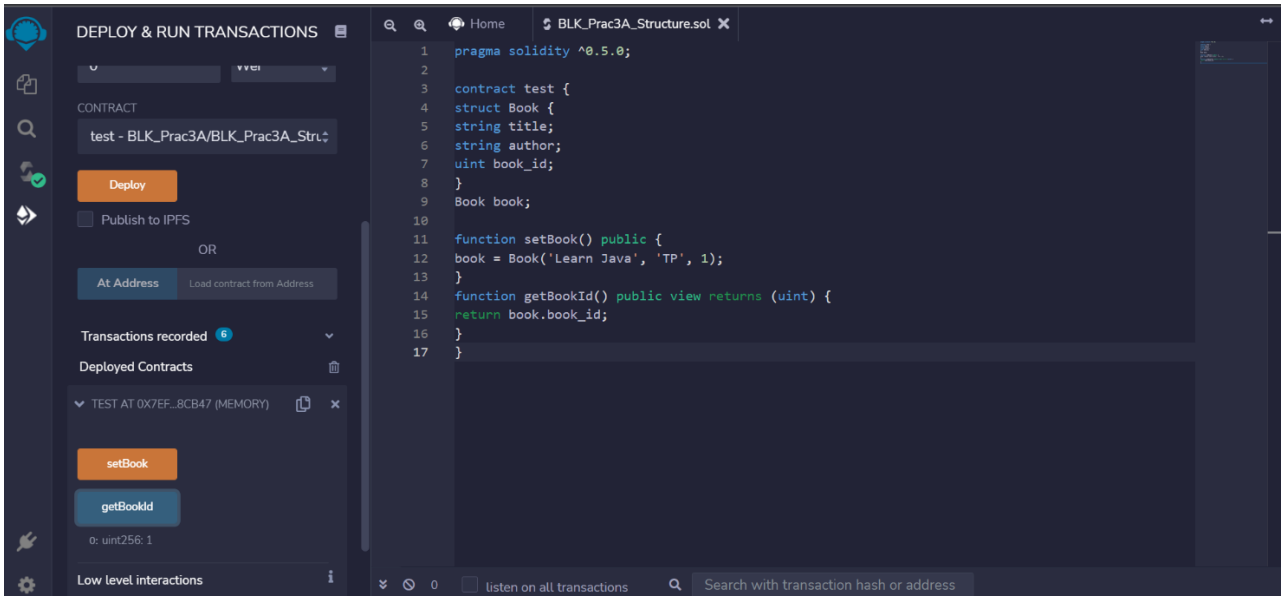


Structure: Struct types are used to represent a record.

Code:

```
ragma solidity ^0.5.0;

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 :



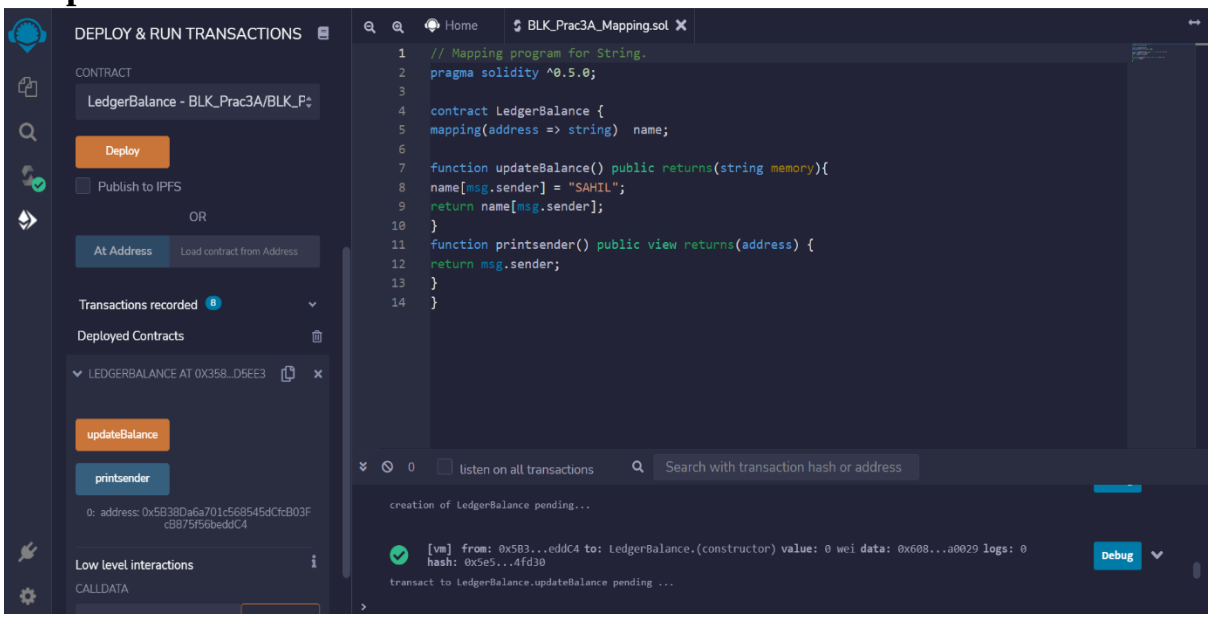
Mappings:

Mapping is a reference type as arrays and structs. Following is the syntax to declare a mapping type. mapping(_KeyType => _ValueType) where ,
_KeyType − can be any built-in types plus bytes and string. No reference type or complex objects are allowed.
_ValueType − can be any type.

Code:

```
// Mapping program for String.
pragma solidity ^0.5.0;

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

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

Output :



PRACTICAL NO. 3

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

Function : A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Code:

```
pragma solidity ^0.5.0; contract SolidityTest {
constructor() public{ }
function getResult() public view returns(string memory){ uint a = 1;
uint b = 2;       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);//access local variable       }}
```

Output :

View Function : View functions ensure that they will not modify the state. A function can be declared as view. Getter method are by default view functions.

- Modifying state variables.
- Emitting events.
- Creating other contracts.
- Using self destruct.
- Sending Ether via calls.
- Calling any function which is not marked view or pure.
- Using low-level calls.
- Using inline assembly containing certain opcodes.

Code:

```
pragma solidity ^0.5.0;
   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 :

Pure Function :

Pure functions ensure that they not read or modify the state. A function can be declared as pure. Pure functions can use the revert() and require() functions to revert potential state changes if an error occurs.

- Reading state variables.

- Accessing address(this).balance or <address>.balance.

- Accessing any of the special variable of block, tx, msg (msg.sig and msg.data can be read).

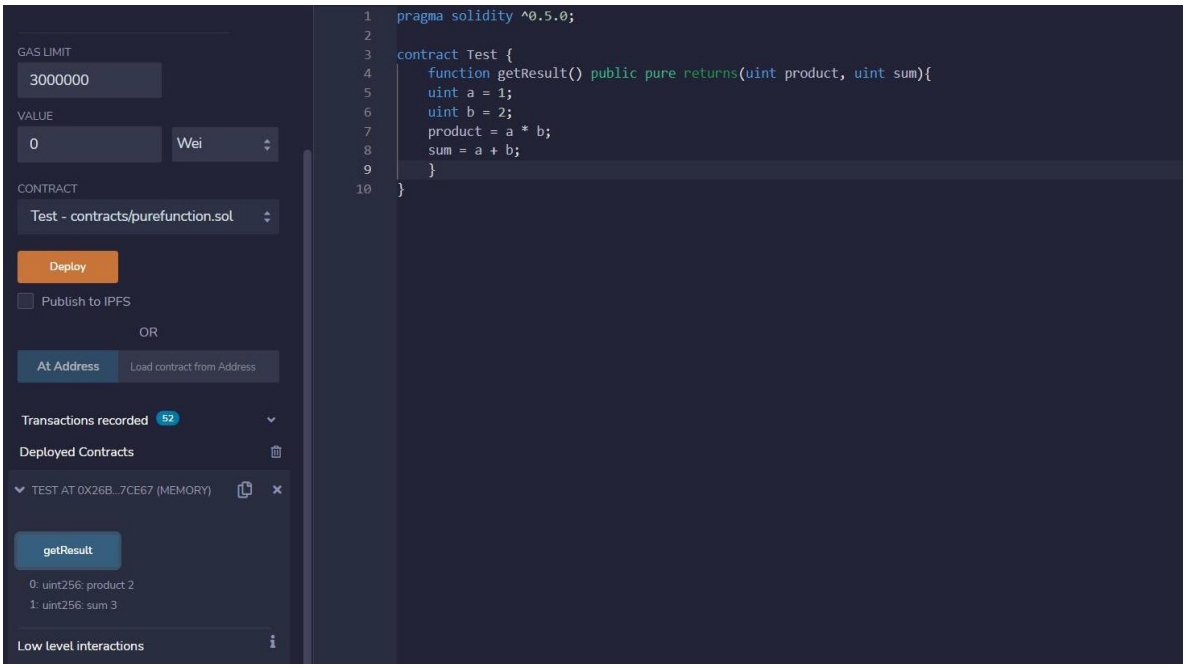
- Calling any function not marked pure.

- Using inline assembly that contains certain opcodes.

Code:

```
pragma solidity ^0.5.0;

contract Test {
    function getResult() public pure returns(uint product, uint sum){
    uint a = 1;
    uint b = 2;
    product = a * b;
    sum = a + b;
    }
}
```

Output :



Fallback Function :

Fallback function is a special function available to a contract.
- It is called when a non-existent function is called on the contract.
- It is required to be marked external.
- It has no name.
- It has no arguments
- It can not return any thing.
- It can be defined one per contract.

If not marked payable, it will throw exception if contract receives plain ether without data.

Code:

```
pragma solidity ^0.5.0;
contract Test {
    uint public x ;
    function() external { x = 1; }        }
contract Sink {
    function() external payable { }
}
contract Caller {
    function callTest(Test test) public returns (bool) {
        (bool success,) = address(test).call(abi.encodeWithSignature("nonExistingFunction()"));
        require(success);
// test.x is now 1
        address payable testPayable = address(uint160(address(test)));
// Sending ether to Test contract,
// the transfer will fail, i.e. this returns false here.
        return (testPayable.send(2 ether));        }
    function callSink(Sink sink) public returns (bool) {
        address payable sinkPayable = address(sink);
        return (sinkPayable.send(2 ether));       }
}
```

Output :

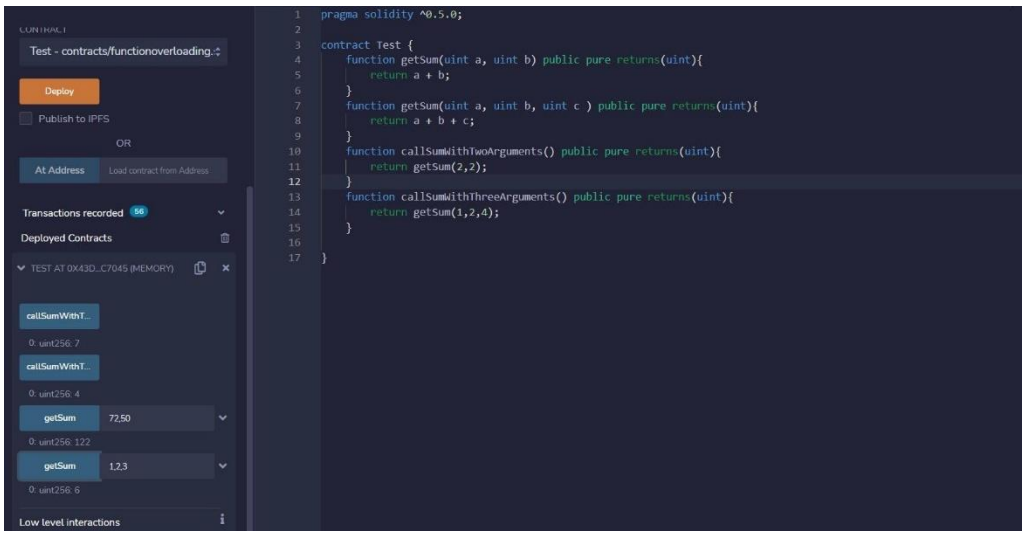
Function Overloading :

The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

Code:

```
pragma solidity ^0.5.0;
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(2,2);
   }
   function callSumWithThreeArguments() public pure returns(uint){
      return getSum(1,2,4);
   }
}
```

Output :



Mathematical Function : Solidity provides inbuilt mathematical functions as well. Following are heavily used methods −

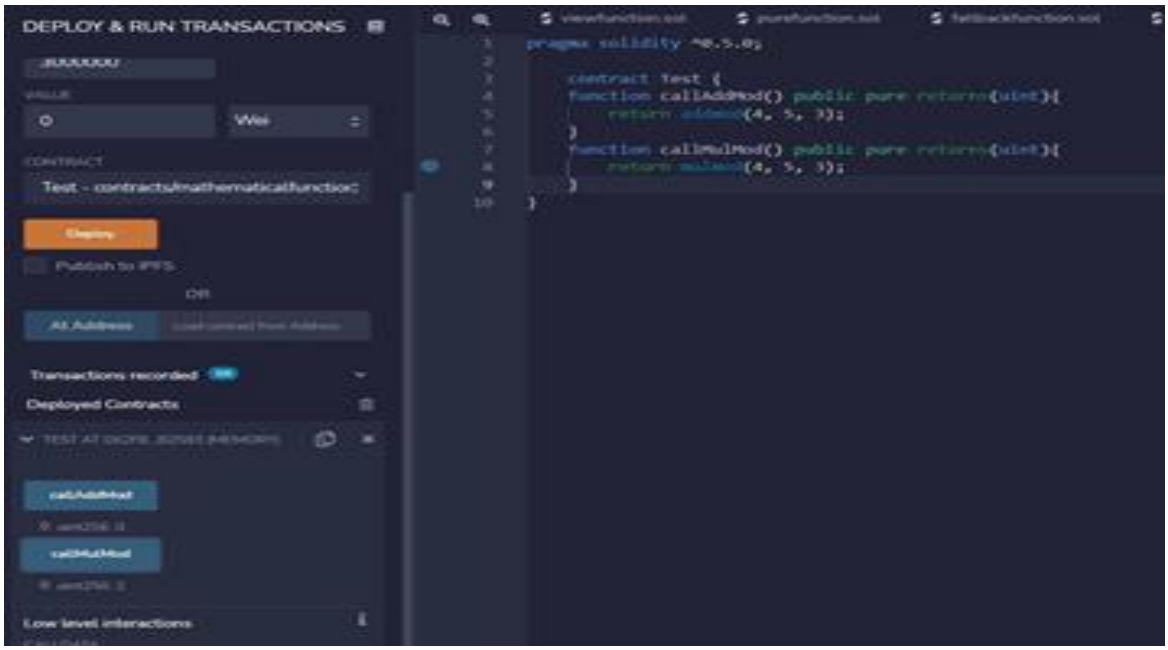
addmod(uint x, uint y, uint k) returns (uint) − computes (x + y) % k where the addition is performed with arbitrary precision and does not wrap around at 2256.

mulmod(uint x, uint y, uint k) returns (uint) − computes (x * y) % k where the addition is performed with arbitrary precision and does not wrap around at 2256.

Code:

```
pragma solidity ^0.5.0;
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 :



Cryptographic Function :

Solidity provides inbuilt cryptographic functions as well.

Following are important methods −

keccak256(bytes memory) returns (bytes32) − computes the Keccak-256 hash of the input.

ripemd160(bytes memory) returns (bytes20) − compute RIPEMD-160 hash of the input.

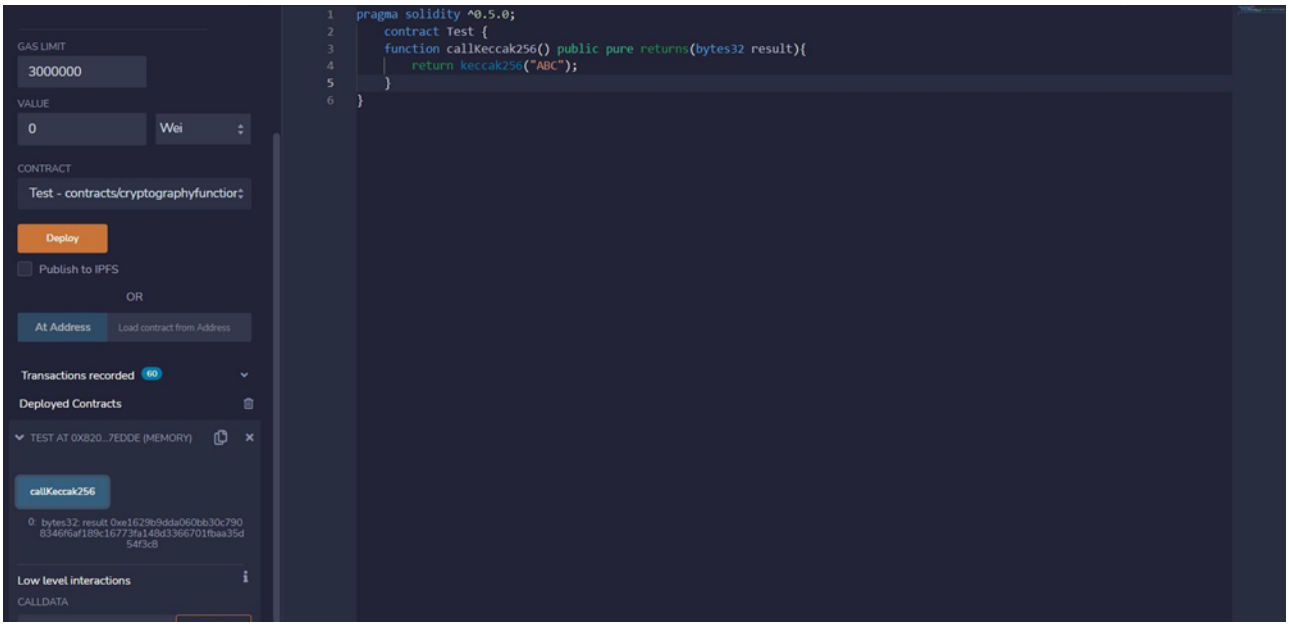
sha256(bytes memory) returns (bytes32) − computes the SHA-256 hash of the input.

ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address) − recover the address associated with the public key from elliptic curve signature or return zero on error. The function parameters correspond to ECDSA values of the signature: r - first 32 bytes of signature; s: second 32 bytes of signature; v: final 1 byte of signature. This method returns an address.

Code:

```
pragma solidity
   ^0.5.0; contract
   Test {
   function callKeccak256() public pure returns(bytes32 result){
      return keccak256("ABC");            }
}
```

Output :



PRACTICAL NO. 4

A] Withdrawal Pattern, Restricted Access.

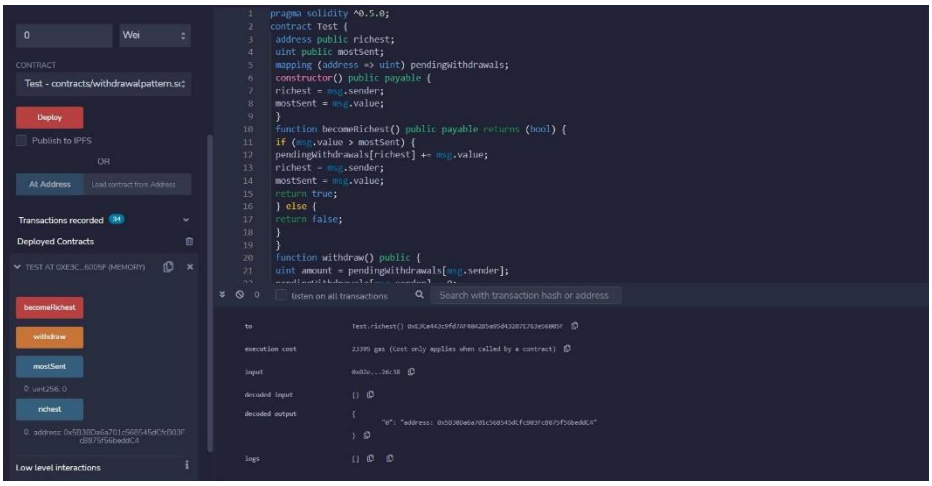
Withdrawal Pattern

Withdrawal pattern ensures that direct transfer call is not made which poses a security threat. Following contract is showing the insecure use of transfer call to send ether.

Code:

```
pragma solidity ^0.5.0; contract Test {
address public richest; uint public mostSent;
mapping (address => uint) pendingWithdrawals; constructor() public payable {
richest = msg.sender; mostSent = msg.value;
}
function becomeRichest() public payable returns (bool) { if (msg.value > mostSent) {
pendingWithdrawals[richest] += msg.value; richest = msg.sender;
mostSent = msg.value;
return true; }
else { return false;
}}
function withdraw() public {
uint amount = pendingWithdrawals[msg.sender]; pendingWithdrawals[msg.sender] = 0;
msg.sender.transfer(amount);
}}
```

Output:



Restricted Access

Restricted Access to a Contract is a common practice. By Default, a contract state is read-only unless it is specified as public.

We can restrict who can modify the contract's state or call a contract's functions using modifiers.

We will create and use multiple modifiers as explained below :

onlyBy − once used on a function then only the mentioned caller can call this function.

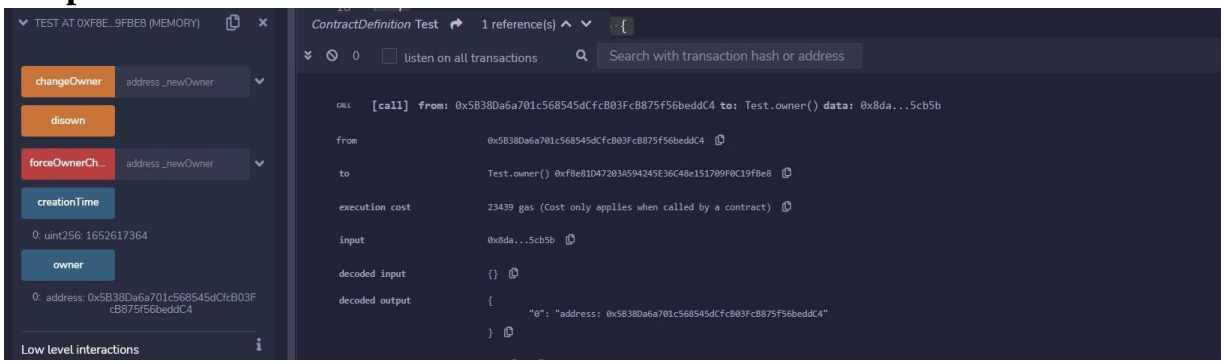
onlyAfter − once used on a function then that function can be called after certain time period.

costs − once used on a function then caller can call this function only if certain value is provided.

Code:

```
pragma solidity ^0.5.0; contract Test {
address public owner = msg.sender; uint public creationTime = now;
modifier onlyBy(address _account) { require(msg.sender == _account, "Sender not
authorized.");  }
function changeOwner(address _newOwner) public onlyBy(owner) { owner = _newOwner; }
modifier onlyAfter(uint _time) { require(now >= _time,"Function called too early.");
_; }
function disown() public onlyBy(owner) onlyAfter(creationTime + 6 weeks) { delete owner;}
modifier costs(uint _amount) { require(
msg.value >= _amount,
"Not enough Ether provided."
);
_;
if (msg.value > _amount) msg.sender.transfer(msg.value - _amount);
}
function forceOwnerChange(address _newOwner) public payable costs(200 ether) {

owner = _newOwner;
if (uint(owner) & 0 == 1) return;
}}
```

Output:



PRACTICAL NO. 4

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

Contracts

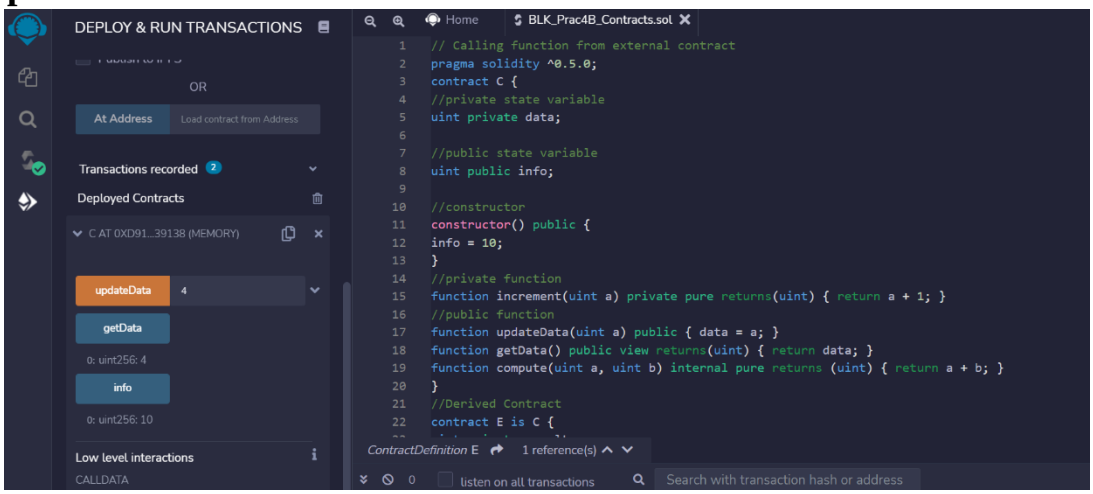
Contract in Solidity is similar to a Class in C++. A Contract have following properties :
Constructor − A special function declared with constructor keyword which will be executed once per contract and is invoked when a contract is created.
State Variables − Variables per Contract to store the state of the contract.
Functions − Functions per Contract which can modify the state variables to alter the state of a contract.

Visibility Quantifiers: Following are various visibility quantifiers for functions/state variables of a contract.
external − External functions are meant to be called by other contracts. They cannot be used for internal call. To call external function within contract this.function_name() call is required. State variables cannot be marked as external.
public − Public functions/ Variables can be used both externally and internally. For public state variable, Solidity automatically creates a getter function.
internal− Internal functions/ Variables can only be used internally or by derived contracts.
private − Private functions/ Variables can only be used internally and not even by derived contracts.

Code:

```
// Calling function from external contract
pragma solidity ^0.5.0;
contract C {
uint private data; //private state variable
uint public info; //public state variable
//constructor
constructor() public {  info = 10;        }
//private function
function increment(uint a) private pure returns(uint) { return a + 1; }
//public function
function updateData(uint a) public { data = a; }
function getData() public view returns(uint) { return data; }
function compute(uint a, uint b) internal pure returns (uint) { return a + b; }          }
//Derived Contract
contract E is C {
uint private result;
C private c;
constructor() public {
c = new C();    }
function getComputedResult() public {
result = compute(3, 5);          }
function getResult() public view returns(uint) { return result; }
function getData() public view returns(uint) { return c.info(); }       }
```

Output:



Inheritance

Inheritance is a way to extend functionality of a contract. Solidity supports both single as well as multiple inheritance.

Solidity supports both single as well as multiple inheritance. Following are the key highlights.

A derived contract can access all non-private members including internal methods and state variables. But using this is not allowed.

Function overriding is allowed provided function signature remains same. In case of difference of output parameters, compilation will fail.

We can call a super contract's function using super keyword or using super contract name.

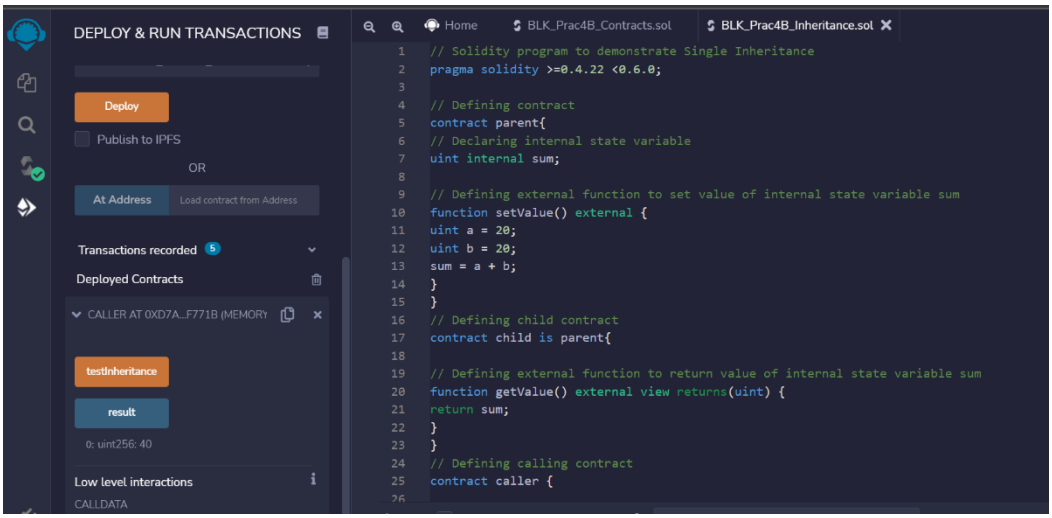
In case of multiple inheritance, function call using super gives preference to most derived contract.

Code:

```
// Solidity program to demonstrate Single Inheritance
pragma solidity >=0.4.22 <0.6.0;
// Defining contract
contract parent{
uint internal sum; // Declaring internal state variable
// Defining external function to set value of internal state variable sum
function setValue() external {
uint a = 20; 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:



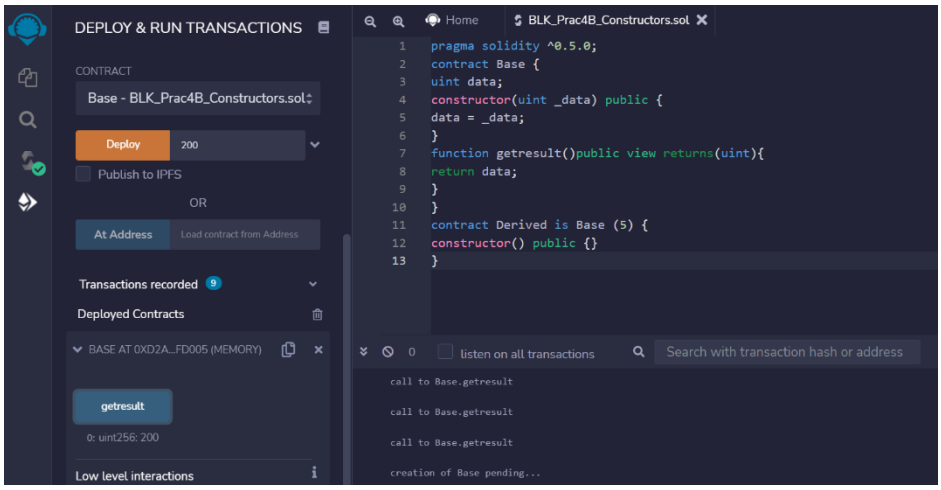
Constructors

Constructor is a special function declared using constructor keyword. It is an optional function and is used to initialize state variables of a contract.
Following are the key characteristics of a constructor.
-A contract can have only one constructor.
-A constructor code is executed once when a contract is created and it is used to initialize contract state.
-A constructor can be either public or internal.
-An internal constructor marks the contract as abstract.
In case, no constructor is defined, a default constructor is present in the contract.

Code:

```
pragma solidity ^0.5.0;
contract Base {
uint data;
constructor(uint _data) public {
data = _data;   }
function getresult()public view returns(uint){
return data;     }         }
contract Derived is Base (5) {
constructor() public {}    }
```

Output:

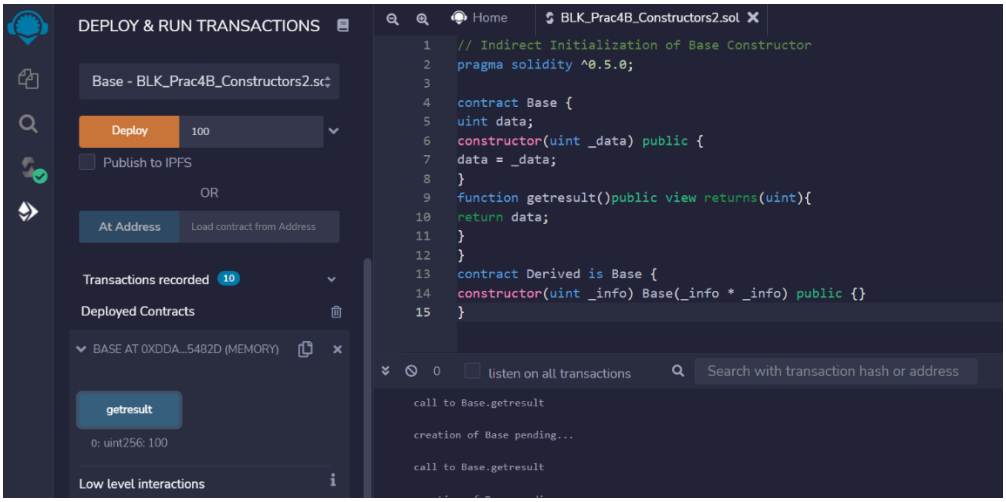


Code:

// Indirect Initialization of Base Constructor

```
pragma solidity ^0.5.0;
contract Base { uint data;
constructor(uint _data) public { data = _data;          }
function getresult()public view returns(uint){ return data; } }
contract Derived is Base {
constructor(uint _info) Base(_info * _info) public {}  }
```

Output:



Abstract Contracts

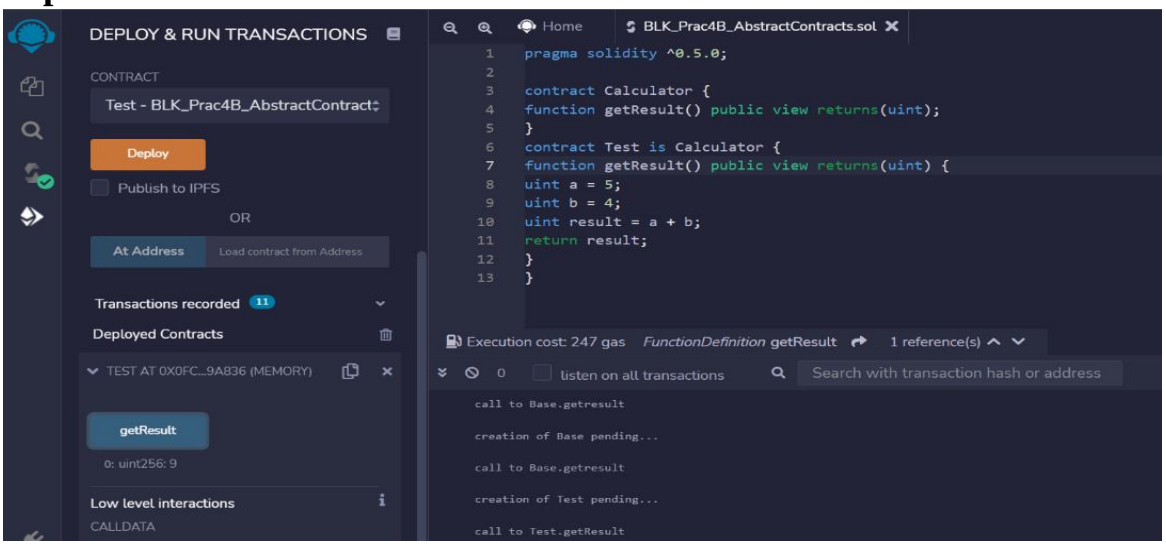
Abstract Contract is one which contains at least one function without any implementation. Such a contract is used as a base contract. Generally, an abstract contract contains both implemented as well as abstract functions. Derived contract will implement the abstract function and use the existing functions as and when required.

Code:

```
pragma solidity ^0.5.0;
contract Calculator {
```

function getResult() public view returns(uint); }
contract Test is Calculator { function getResult() public view returns(uint) {
uint a = 5; uint b = 4; uint result = a + b; return result; }}

Output:



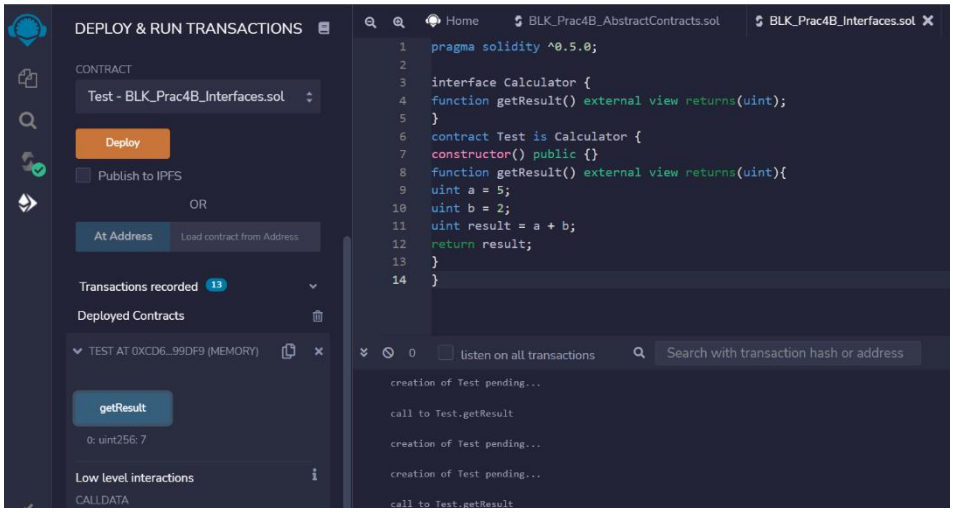
Interfaces

Interfaces are similar to abstract contracts and are created using interface keyword. Following are the key characteristics of an interface.
-Interface cannot have any function with implementation.
-Functions of an interface can be only of type external.
-Interface cannot have constructor.
-Interface cannot have state variables.

Code:

pragma solidity ^0.5.0;
interface Calculator { function getResult() external view returns(uint); }
contract Test is Calculator { constructor() public {}
function getResult() external view returns(uint){ uint a = 5; uint b = 3; uint result = a + b;
return result; }}

Output:



PRACTICAL NO. 4

C] Libraries, Assembly, Events, Error handling.

Libraries

Libraries are similar to Contracts but are mainly intended for reuse. A Library contains functions which other contracts can call. Solidity have certain restrictions on use of a Library. Following are the key characteristics of a Solidity Library.

-Library functions can be called directly if they do not modify the state. That means pure or view functions only can be called from outside the library.

-Library can not be destroyed as it is assumed to be stateless.

-A Library cannot have state variables.

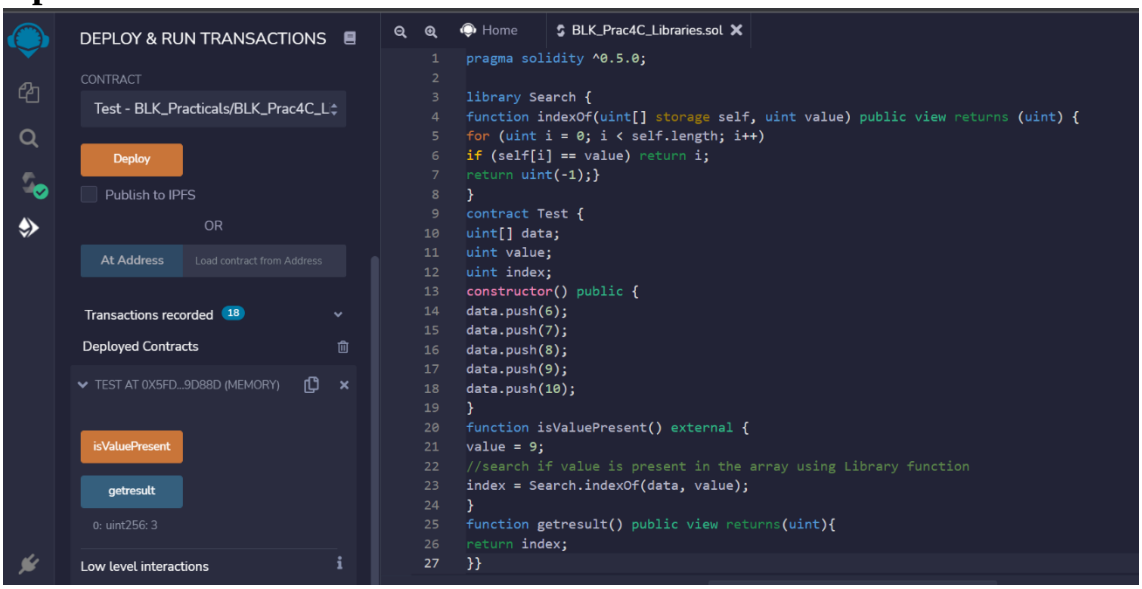
-A Library cannot inherit any element.

-A Library cannot be inherited.

Code:

```
pragma solidity ^0.5.0;
library Search {
function indexOf(uint[] storage self, uint value) public view returns (uint) {
for (uint i = 0; i < self.length; i++)
if (self[i] == value) return i;
return uint(-1);}          }
contract Test { uint[] data; uint value; uint index;
constructor() public {
data.push(6); data.push(7); data.push(8); data.push(9); data.push(10);                    }
function isValuePresent() external {
value = 9;
//search if value is present in the array using Library function
index = Search.indexOf(data, value);          }
function getresult() public view returns(uint){  return index;          }}
```

Output:



Assembly

Solidity provides an option to use assembly language to write inline assembly within Solidity source code. We can also write a standalone assembly code which then be converted to bytecode. Standalone Assembly is an intermediate language for a Solidity compiler and it converts the Solidity code into a Standalone Assembly and then to byte code. We can used the same language used in Inline Assembly to write code in a Standalone assembly.

Inline Assembly

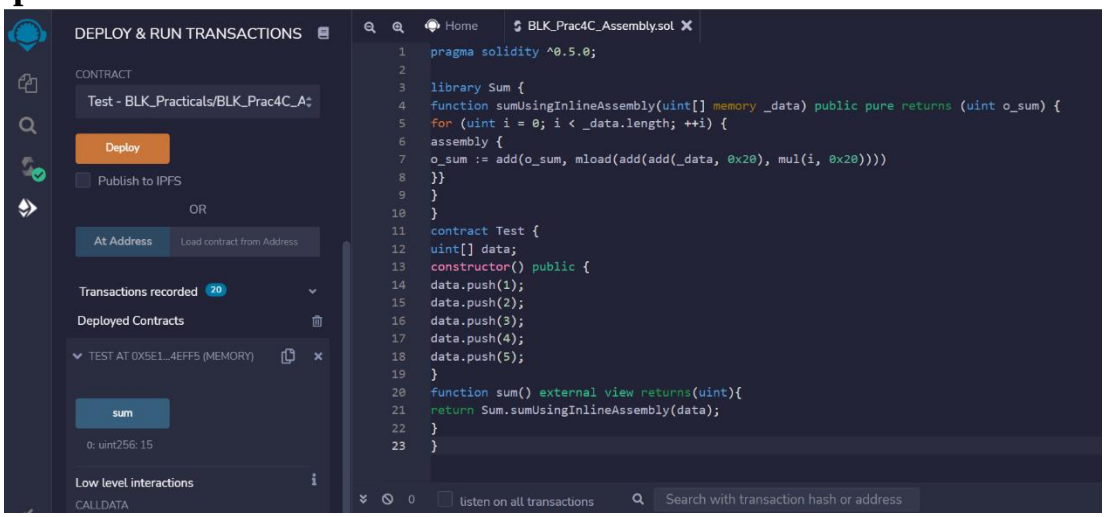
Inline assembly code can be interleaved within Solidity code base to have more fine-grain control over EVM and is used especially while writing the library functions.

An assembly code is written under assembly { ... } block.

Code:

```
pragma solidity ^0.5.0;
library Sum {
function sumUsingInlineAssembly(uint[] memory _data) public pure returns (uint o_sum) {
for (uint i = 0; i < _data.length; ++i) {
assembly { o_sum := add(o_sum, mload(add(add(_data, 0x20), mul(i, 0x20))))      }}
}}
contract Test { uint[] data;
constructor() public {
data.push(1); data.push(2); data.push(3); data.push(4); data.push(5);          }
function sum() external view returns(uint){
return Sum.sumUsingInlineAssembly(data);          }}
```

Output:



Events

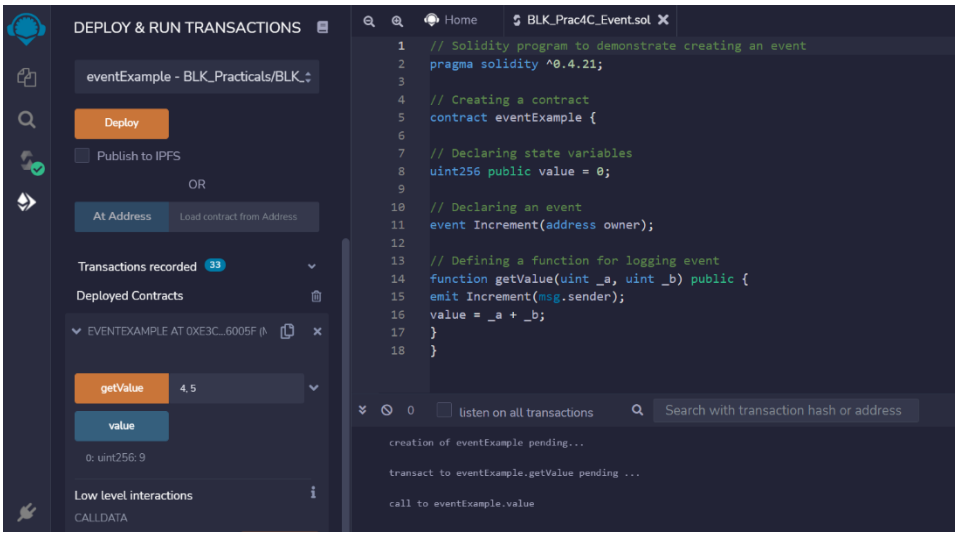
Event is an inheritable member of a contract. An event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain. An event generated is not accessible from within contracts, not even the one which have created and emitted them.

Code:

```
// Solidity program to demonstrate creating an event
```

```
pragma solidity ^0.4.21;
// Creating a contract
contract eventExample { uint256 public value = 0; // Declaring state variables
event Increment(address owner); // Declaring an event
// Defining a function for logging event
function getValue(uint _a, uint _b) public {
emit Increment(msg.sender);
value = _a + _b;          }}
```

Output:



Error Handling

Solidity provides various functions for error handling. Generally when an error occurs, the state is reverted back to its original state. Other checks are to prevent unauthorized code access.
Following are some of the important methods used in error handling −
assert(bool condition) − In case condition is not met, this method call causes an invalid opcode and any changes done to state got reverted. This method is to be used for internal errors.
require(bool condition) − In case condition is not met, this method call reverts to original state. - This method is to be used for errors in inputs or external components.
require(bool condition, string memory message) − In case condition is not met, this method call reverts to original state. - This method is to be used for errors in inputs or external components. It provides an option to provide a custom message.
revert() − This method aborts the execution and revert any changes done to the state.
revert(string memory reason) − This method aborts the execution and revert any changes done to the state. It provides an option to provide a custom message.

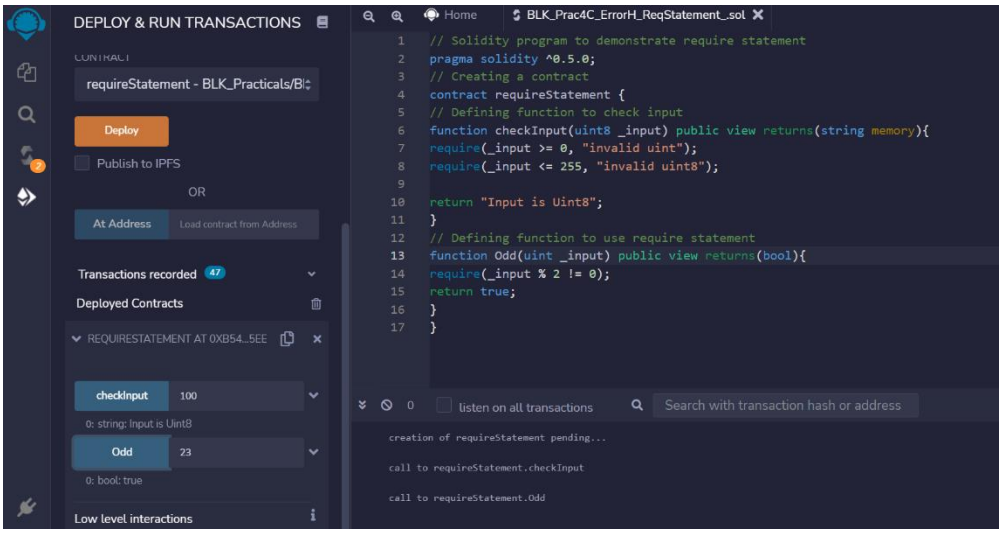
require statement

Code:

```
// Solidity program to demonstrate require statement
pragma solidity ^0.5.0;
// Creating a contract
contract requireStatement {
// Defining function to check input
```

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

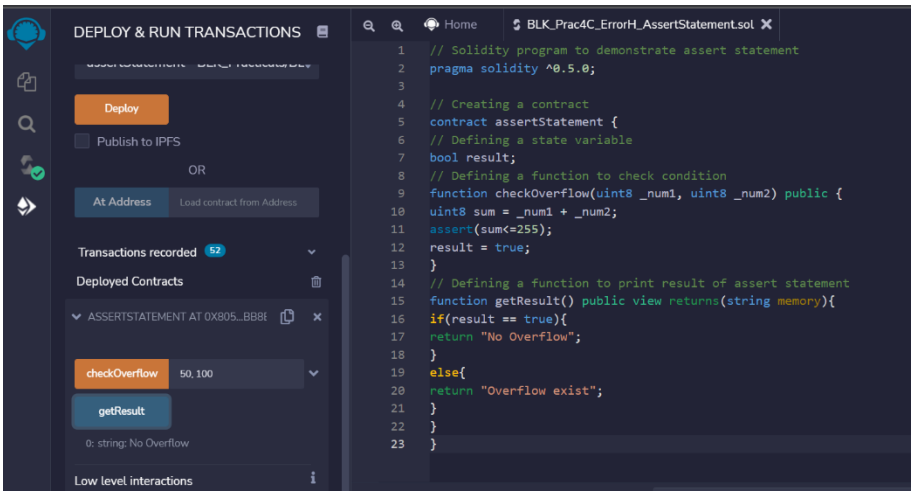


assert statement

Code:

```
// Solidity program to demonstrate assert statement
pragma solidity ^0.5.0;
// Creating a contract
contract assertStatement {
bool result; // Defining a state variable
// Defining a function to check condition
function checkOverflow(uint8 _num1, uint8 _num2) public {
uint8 sum = _num1 + _num2;
assert(sum<=255);
result = true;    }
// Defining a function to print result of assert statement
function getResult() public view returns(string memory){
if(result == true){
return "No Overflow";          }
else{
return "Overflow exist";        }}}
```

Output:

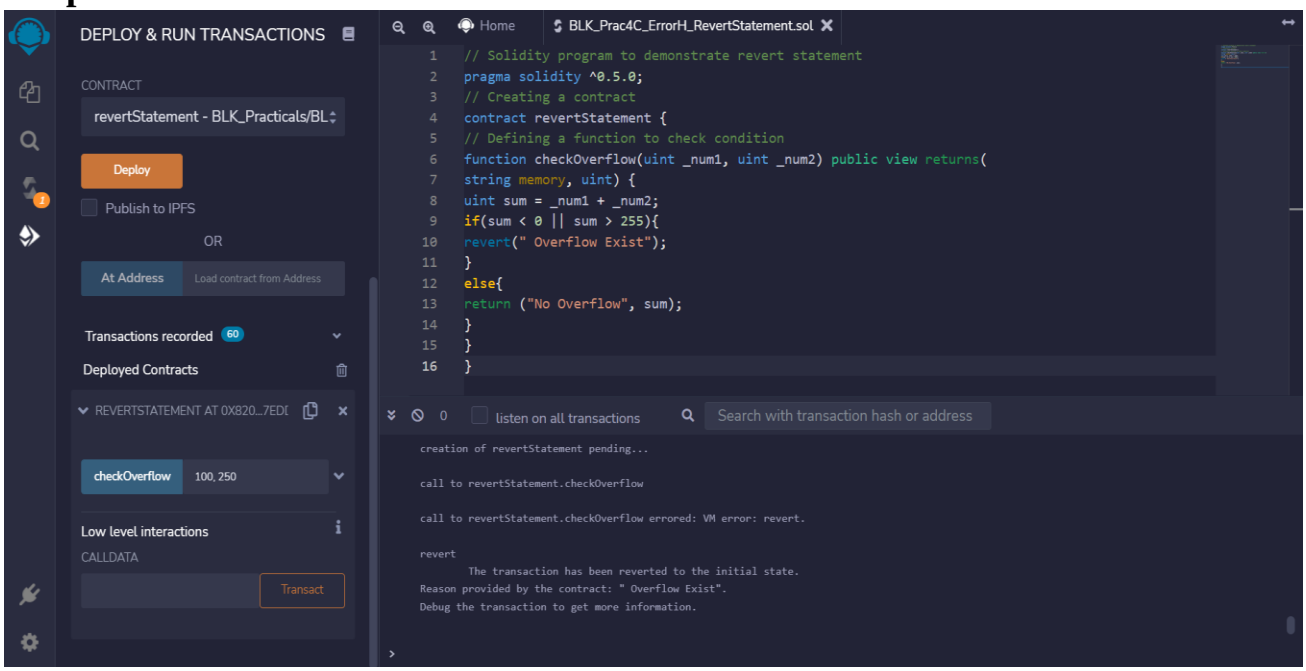


revert statement

Code:

```
// Solidity program to demonstrate revert statement
pragma solidity ^0.5.0;
// Creating a contract
contract revertStatement {
// Defining a function to check condition
function checkOverflow(uint _num1, uint _num2) public view returns(string memory, uint) {
uint sum = _num1 + _num2;
if(sum < 0 || sum > 255){
revert(" Overflow Exist");      }
else{ return ("No Overflow", sum);    }}}
```

Output:



PRACTICAL NO. 5

Aim : Install Hyperledger fabric & composer. Deploy and execute the application.

Theory :

Hyperledger is an open source collaborative effort created to advance blockchain technologies. The Hyperledger organization has a number of projects for various blockchain solutions, such as smart contract engines, permissioned networks, querying for information inside a ledger, etc.

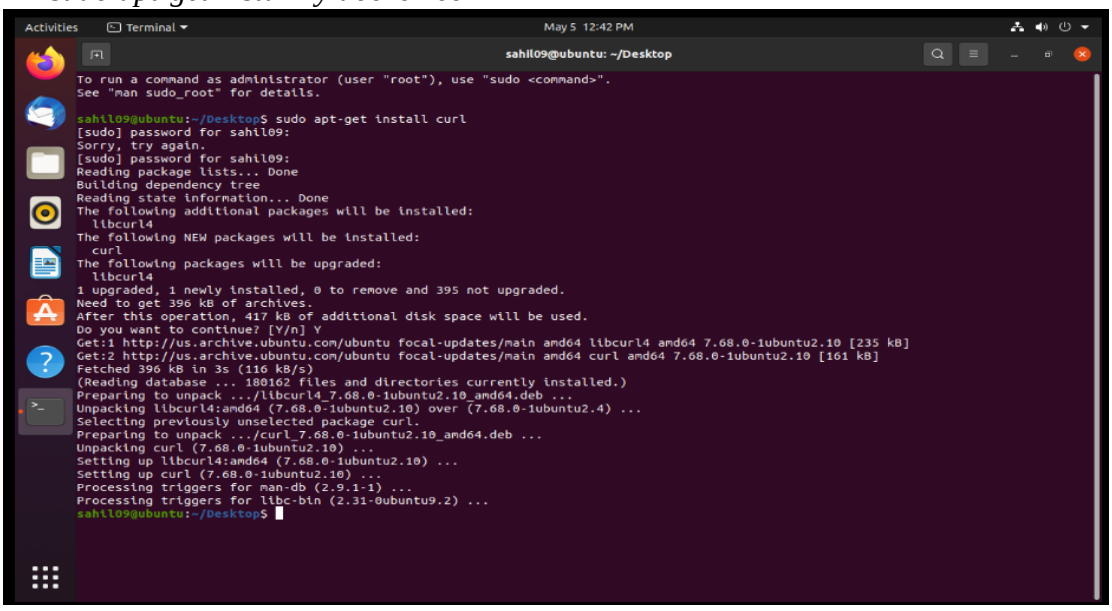
1.*Asset* - Any countable commodity or value. It could be currency, like $US dollars$, Euros, bitcoins. It could be sheep, corn, windows. It can be anything that can be counted.
2.*Account* - An entity that's able to perform a specified set of actions.
3.*Domain* - Assets and accounts are labeled by a domain, which is just a way of grouping them.
4.*Permission* - Does an account have the privilege do perform a certain command?
5.*Role* - A group of permissions.
6.*Transaction* - A set of commands that are applied to the ledger.
7.*Query* - Checking the state of data in the system.

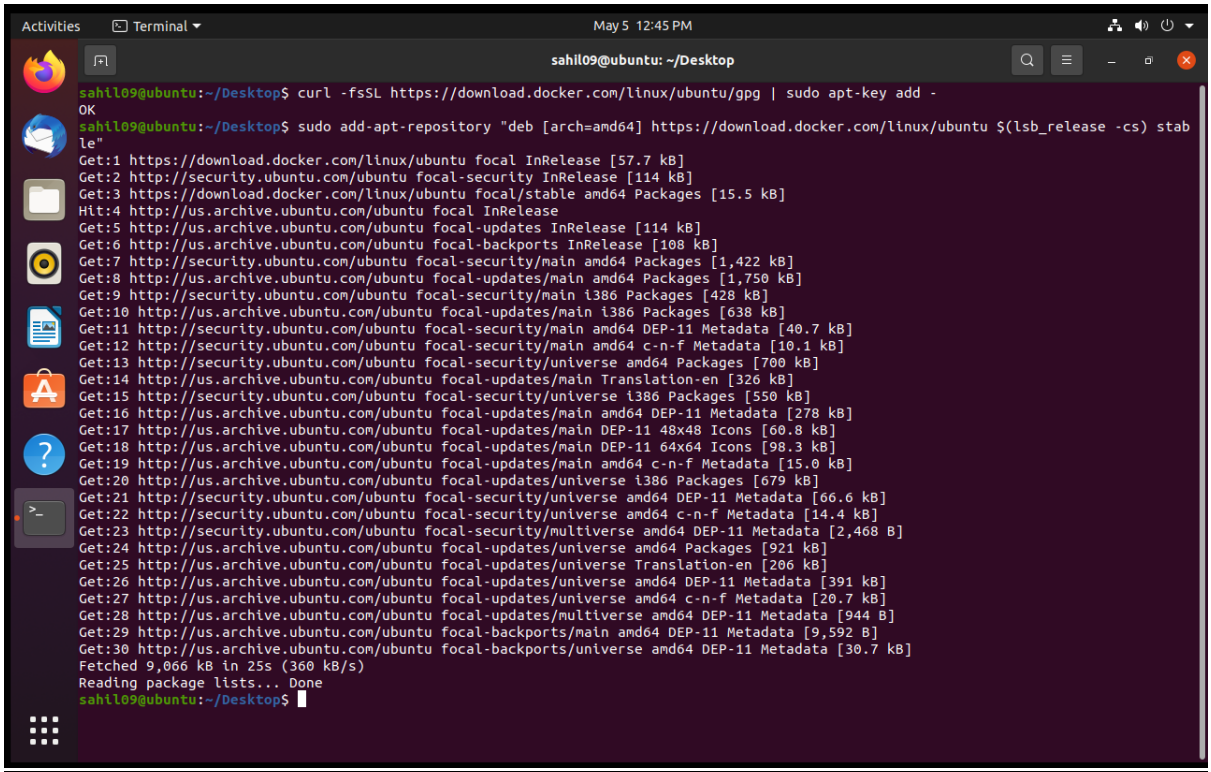
Steps :

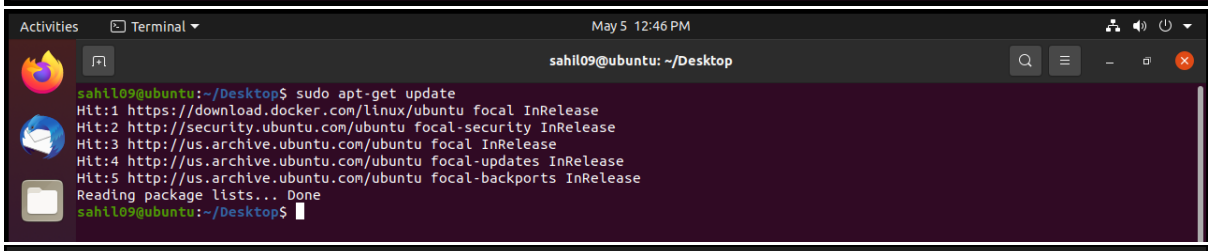
1. Creating an Iroha Network

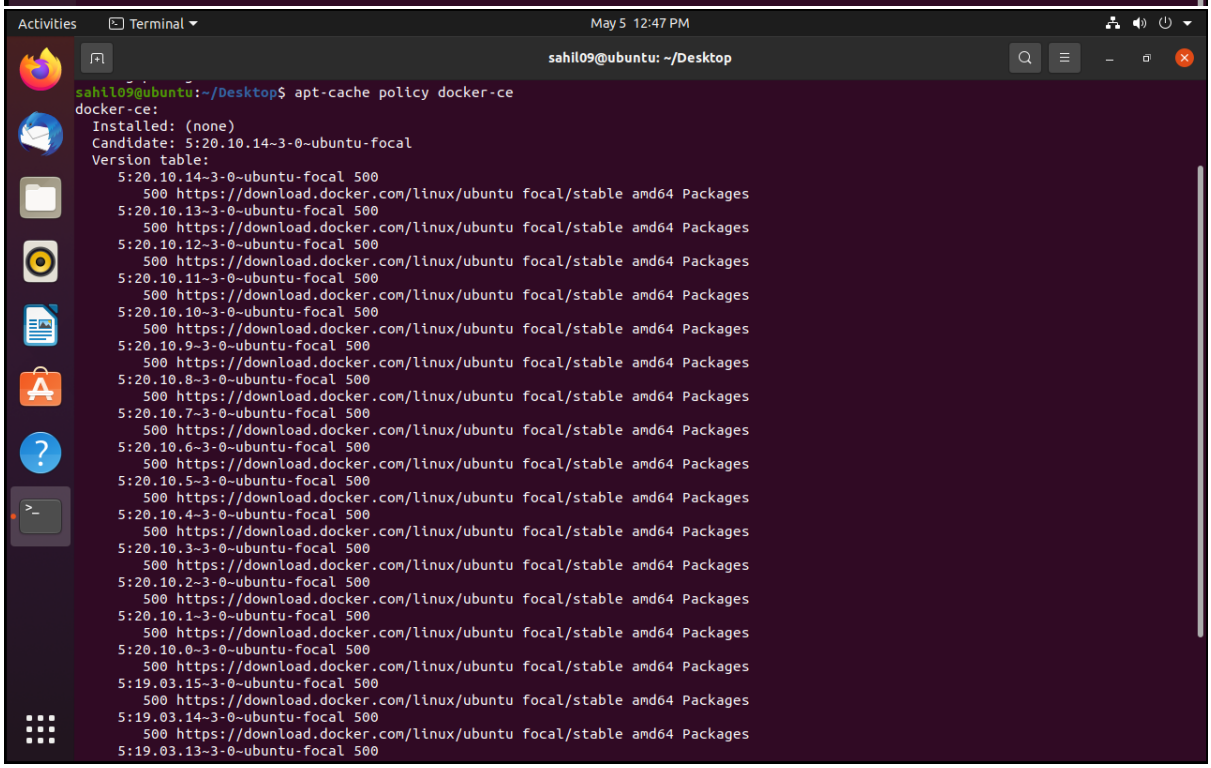
Step 1: Install docker

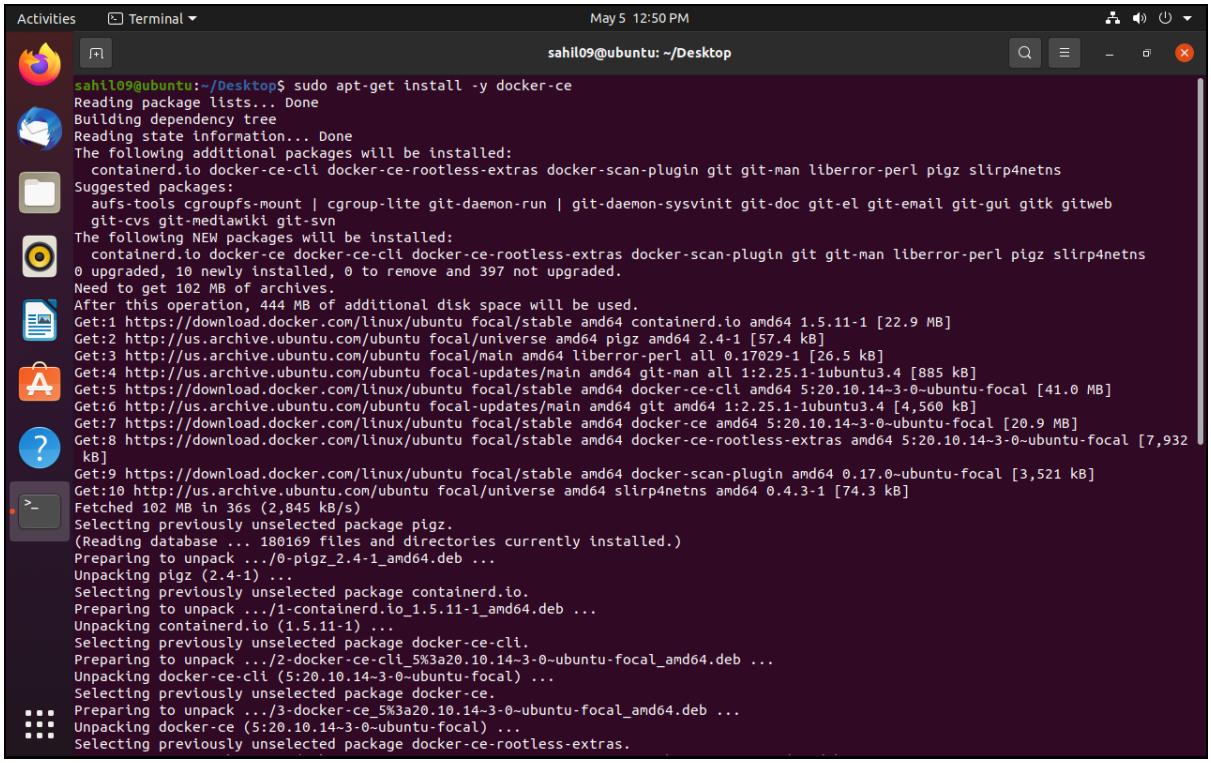
- *sudo apt-get install curl*
- *curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -*
- *sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"*
- *sudo apt-get update*
- *apt-cache policy docker-ce*
- *sudo apt-get install -y docker-ce*



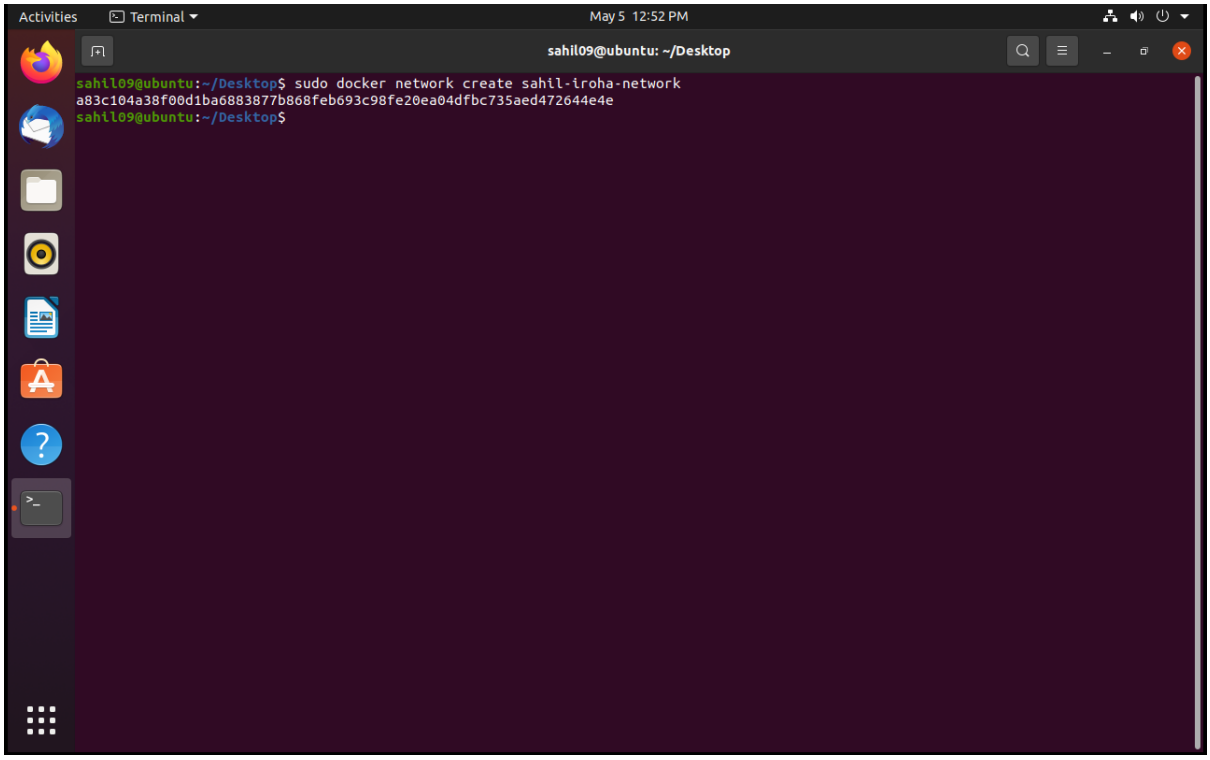

Activities
Terminal
May 5 12:45 PM
sahil09@ubuntu: ~/Desktop
sahil09@ubuntu:~/Desktop$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
sahil09@ubuntu:~/Desktop$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Get:1 https://download.docker.com/linux/ubuntu focal InRelease [57.7 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:3 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [15.5 kB]
Hit:4 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:5 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [1,422 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1,750 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/main i386 Packages [428 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu focal-updates/main i386 Packages [638 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [40.7 kB]
Get:12 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [10.1 kB]
Get:13 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [700 kB]
Get:14 http://us.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [326 kB]
Get:15 http://security.ubuntu.com/ubuntu focal-security/universe i386 Packages [550 kB]
Get:16 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [278 kB]
Get:17 http://us.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 48x48 Icons [60.8 kB]
Get:18 http://us.archive.ubuntu.com/ubuntu focal-updates/main DEP-11 64x64 Icons [98.3 kB]
Get:19 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [15.0 kB]
Get:20 http://us.archive.ubuntu.com/ubuntu focal-updates/universe i386 Packages [679 kB]
Get:21 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [66.6 kB]
Get:22 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-n-f Metadata [14.4 kB]
Get:23 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 Metadata [2,468 B]
Get:24 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [921 kB]
Get:25 http://us.archive.ubuntu.com/ubuntu focal-updates/universe Translation-en [206 kB]
Get:26 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 DEP-11 Metadata [391 kB]
Get:27 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [20.7 kB]
Get:28 http://us.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 DEP-11 Metadata [944 B]
Get:29 http://us.archive.ubuntu.com/ubuntu focal-backports/main amd64 DEP-11 Metadata [9,592 B]
Get:30 http://us.archive.ubuntu.com/ubuntu focal-backports/universe amd64 DEP-11 Metadata [30.7 kB]
Fetched 9,066 kB in 25s (360 kB/s)
Reading package lists... Done
sahil09@ubuntu:~/Desktop$


Activities
Terminal
May 5 12:46 PM
sahil09@ubuntu: ~/Desktop
sahil09@ubuntu:~/Desktop$ sudo apt-get update
Hit:1 https://download.docker.com/linux/ubuntu focal InRelease
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu focal InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:5 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease
Reading package lists... Done
sahil09@ubuntu:~/Desktop$

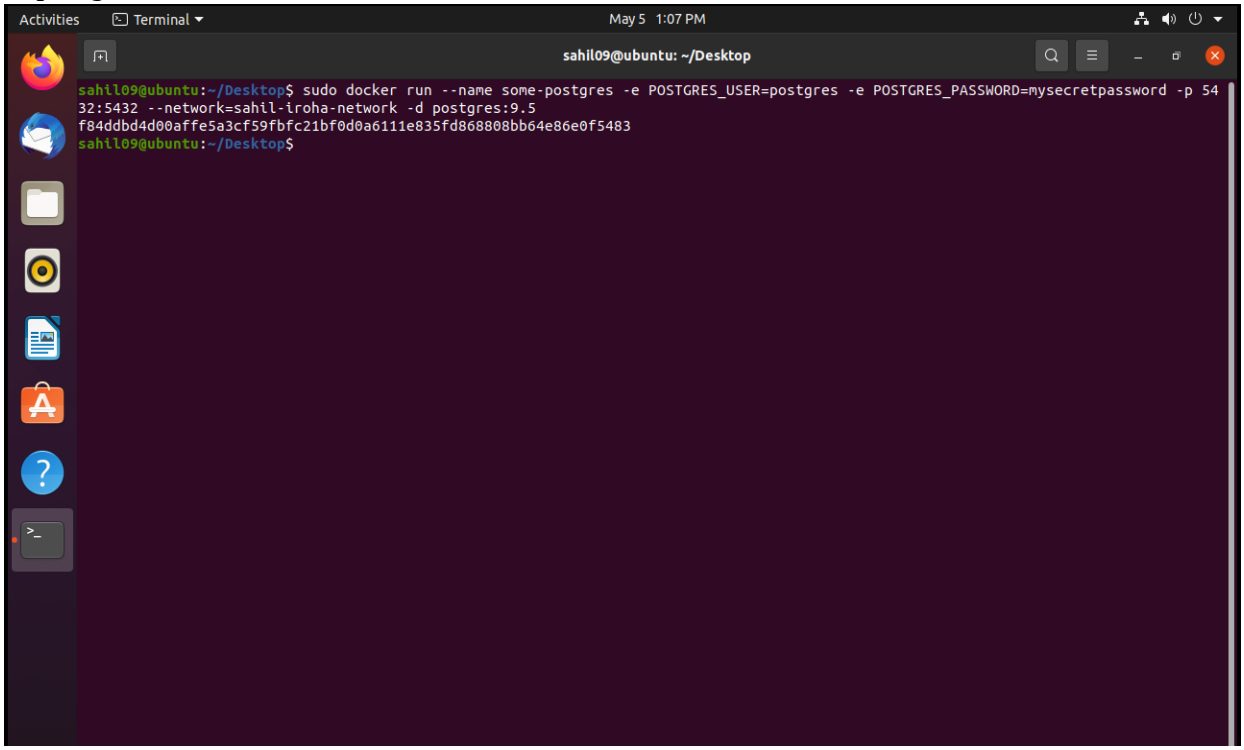

Activities
Terminal
May 5 12:47 PM
sahil09@ubuntu: ~/Desktop
sahil09@ubuntu:~/Desktop$ apt-cache policy docker-ce
docker-ce:
Installed: (none)
Candidate: 5:20.10.14~3-0~ubuntu-focal
Version table:
5:20.10.14~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.13~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.12~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.11~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.10~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.9~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.8~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.7~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.6~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.5~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.4~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.3~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.2~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.1~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:20.10.0~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:19.03.15~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:19.03.14~3-0~ubuntu-focal 500
500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
5:19.03.13~3-0~ubuntu-focal 500



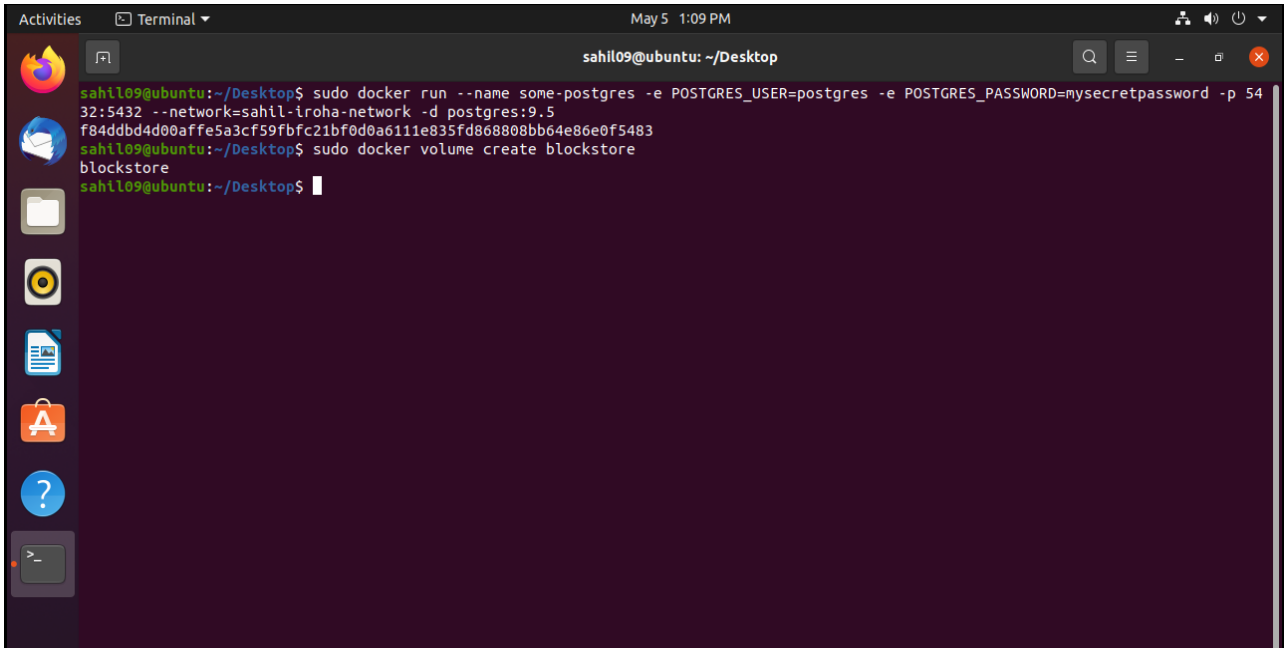
Step 2: Create a docker network. We'll name it "sahil-iroha-network".
sudo docker network create sahil-iroha-network



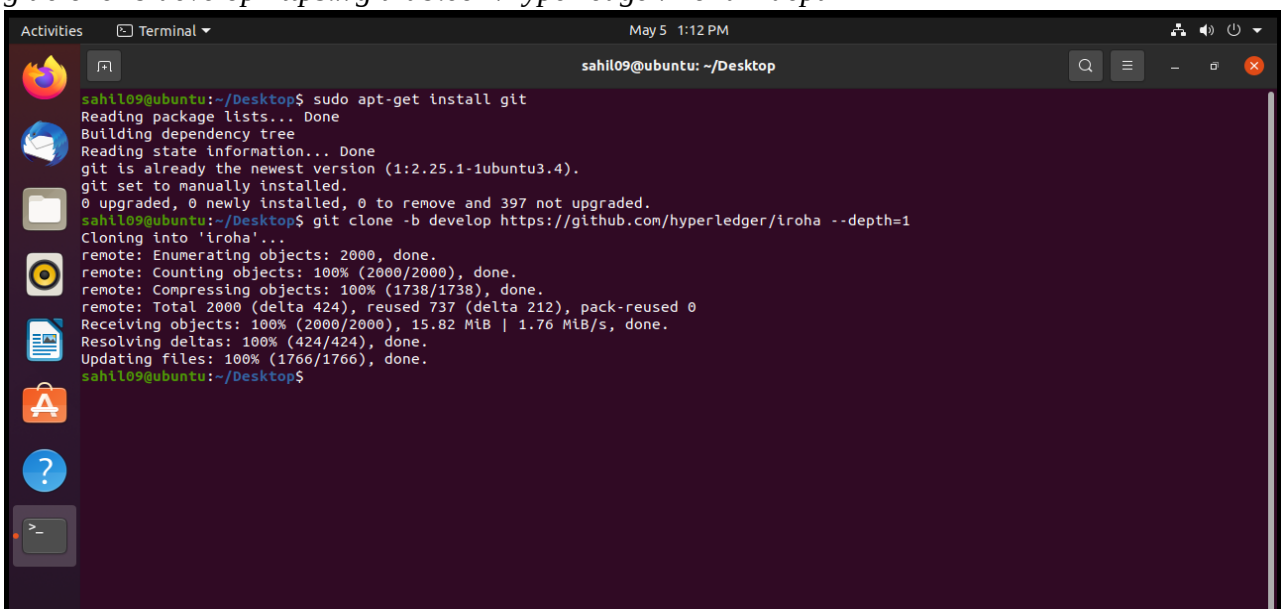
Step 3: Add PostgreSQL to our network.
*sudo docker run --name some-postgres *
*-e POSTGRES_USER=postgres *
*-e POSTGRES_PASSWORD=mysecretpassword *
*-p 5432:5432 *
*--network=srcmake-iroha-network *
-d postgres:9.5



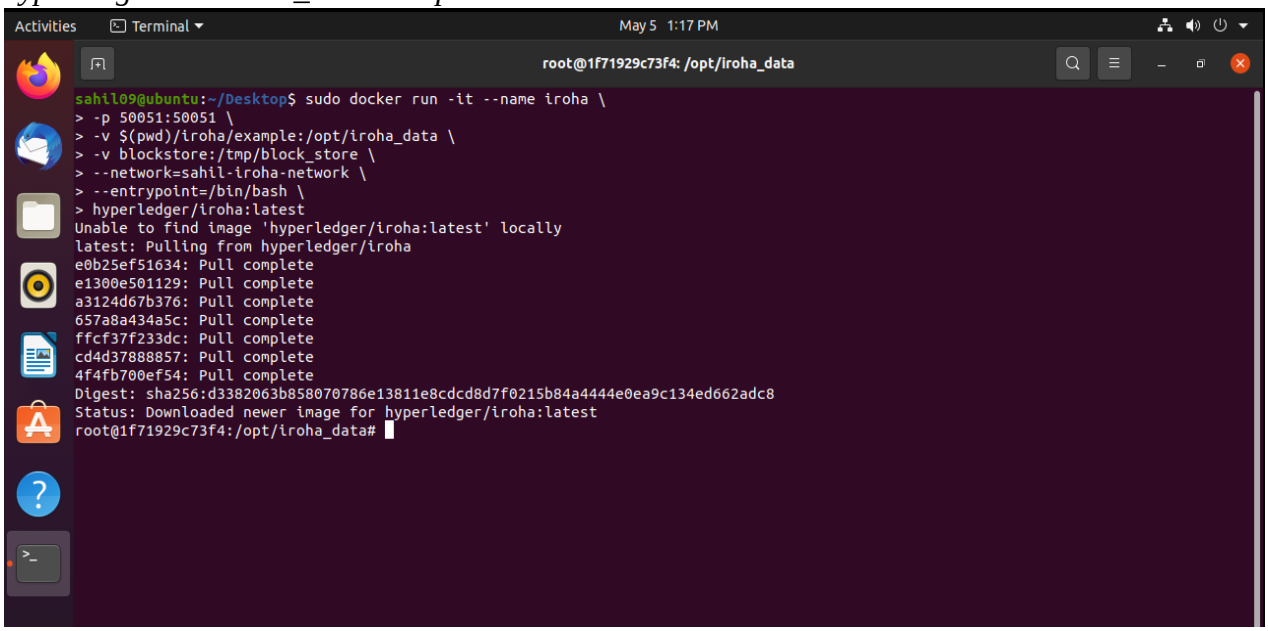
Step 4: Create a volume of persistant storage named "blockstore" to store the blocks for our blockchain.
sudo docker volume create blockstore



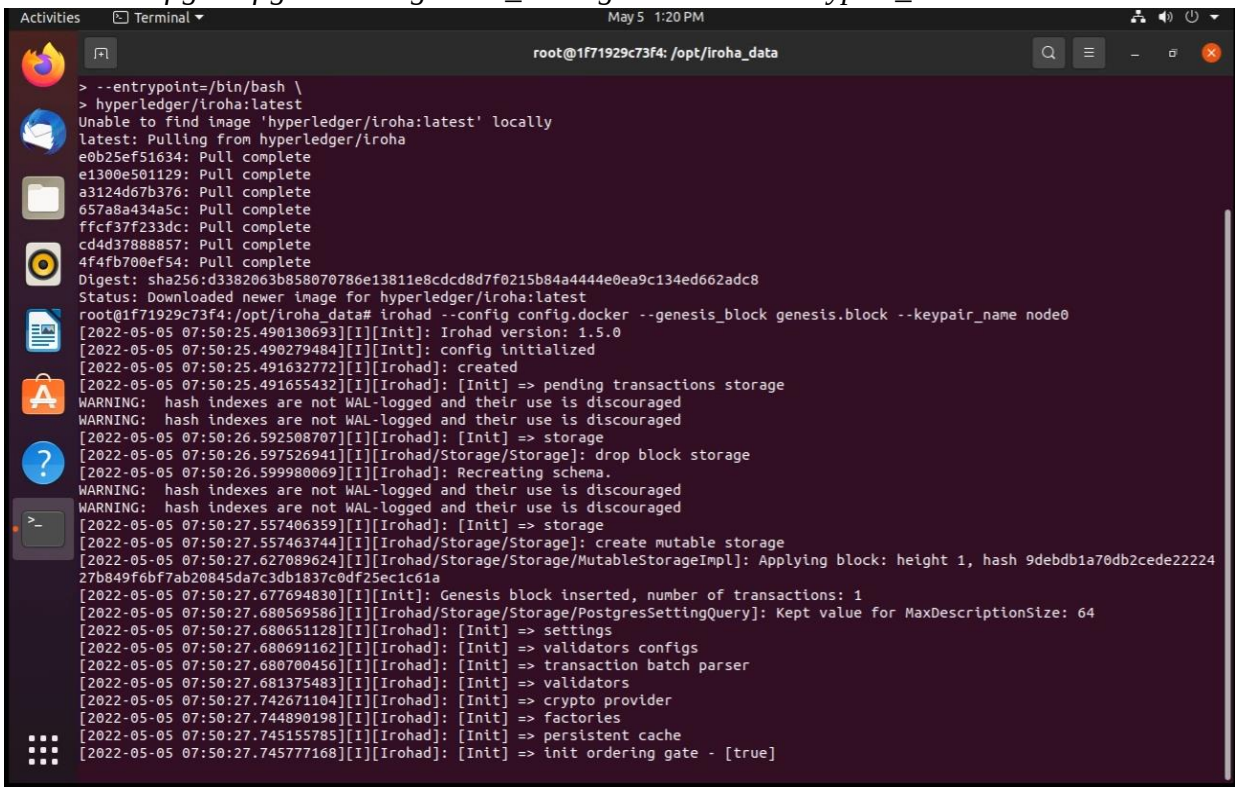
Step 5: Configure Iroha on the network. Download the Iroha code from github. (And install git if you don't already have it.)
sudo apt-get install git
git clone -b develop https://github.com/hyperledger/iroha --depth=1



Step 6: Use the Iroha configuration that's been prepared as an example to run the Iroha docker container.
*sudo docker run -it --name iroha *
*-p 50051:50051 *
*-v $(pwd)/iroha/example:/opt/iroha_data *
*-v blockstore:/tmp/block_store *
*--network=srcmake-iroha-network *
*--entrypoint=/bin/bash *
hyperledger/iroha:x86_64-develop-latest

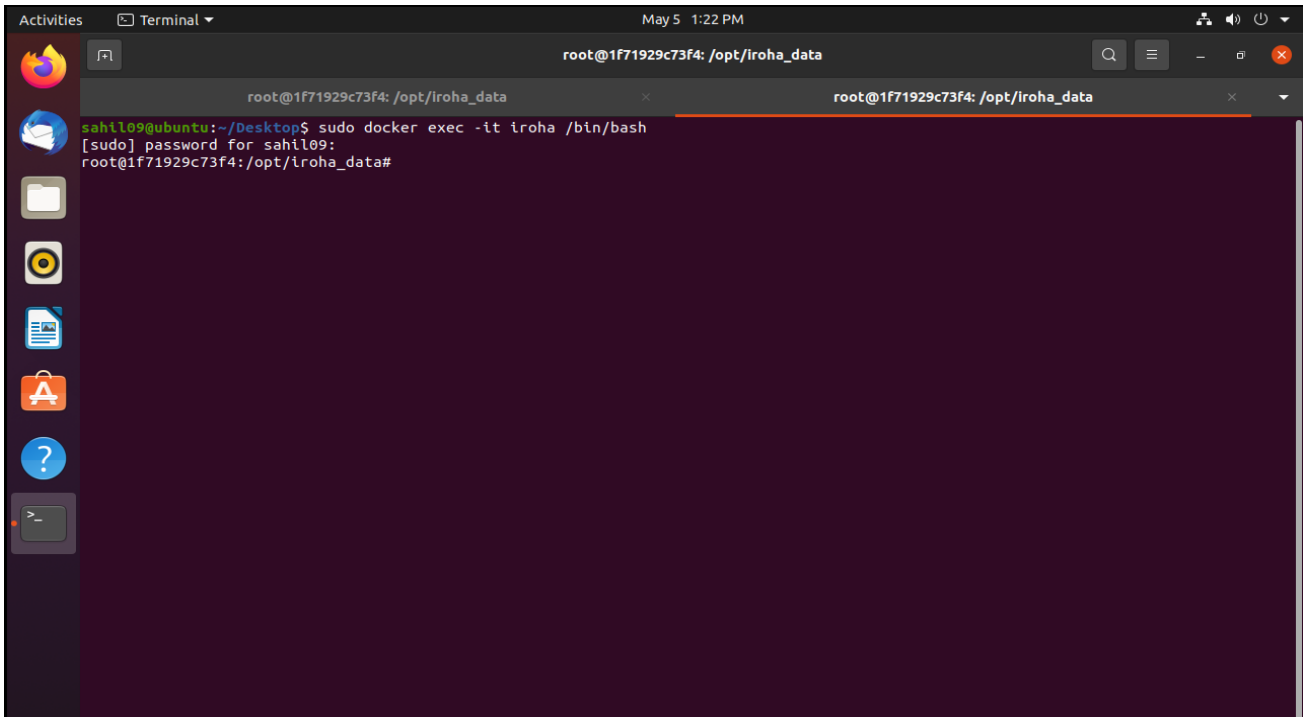


Step 7: Now we're going to actually run Iroha. And now our Iroha blockchain is running in the terminal, so don't close it! Next, we'll do some transactions.
irohad --config config.docker --genesis_block genesis.block --keypair_name node0

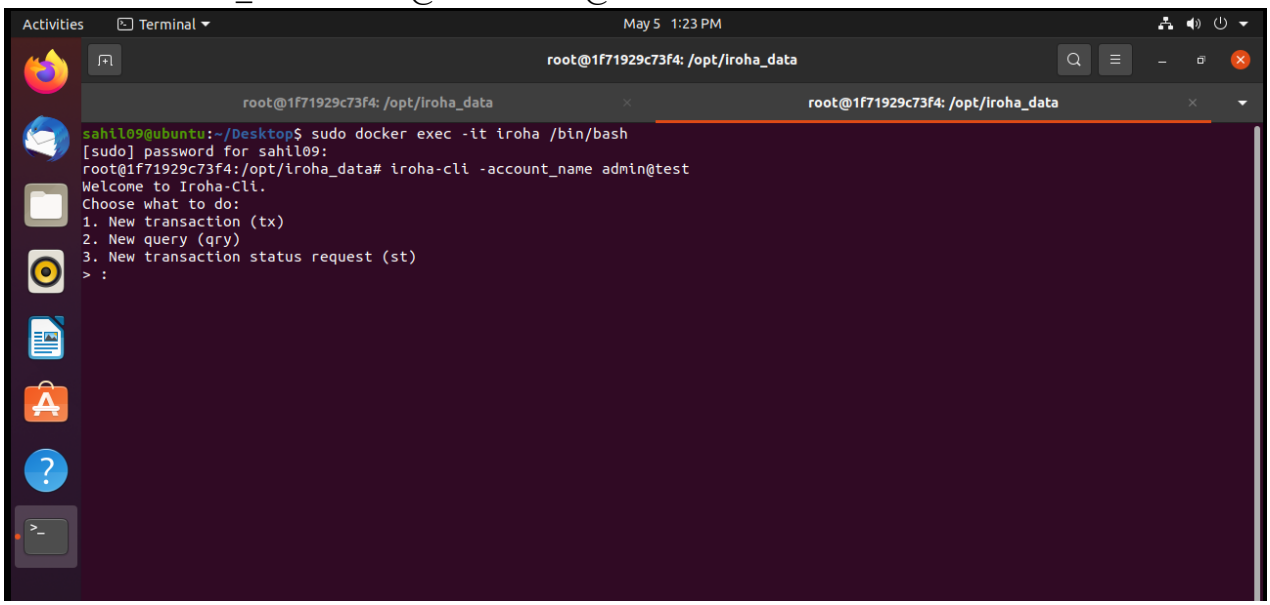


2. Interacting with Iroha Network "sahil-iroha-network".

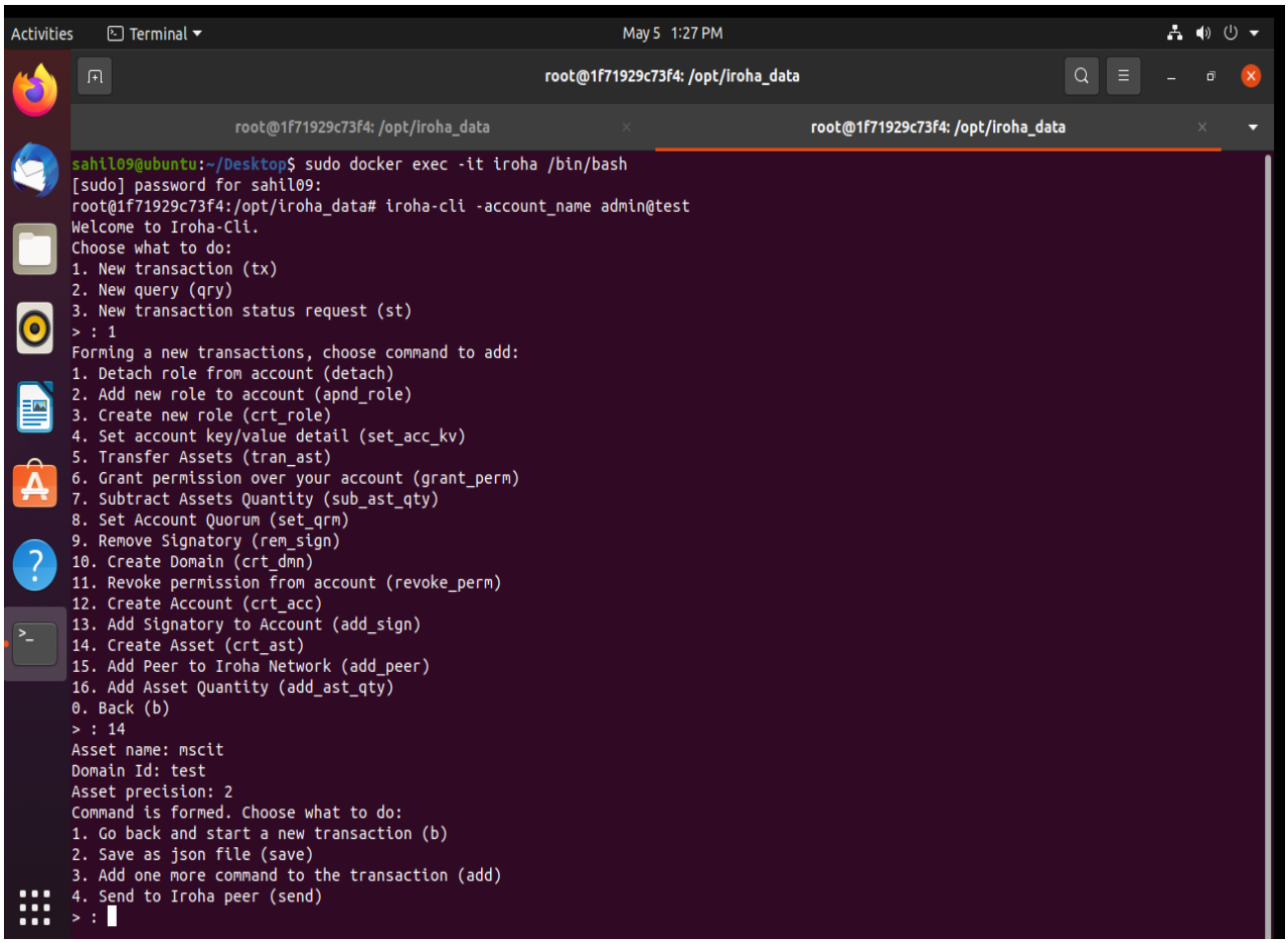
Step 8: Open a new terminal (don't close the one with our Iroha network!) and attach the docker container to our terminal.
sudo docker exec -it iroha /bin/bash



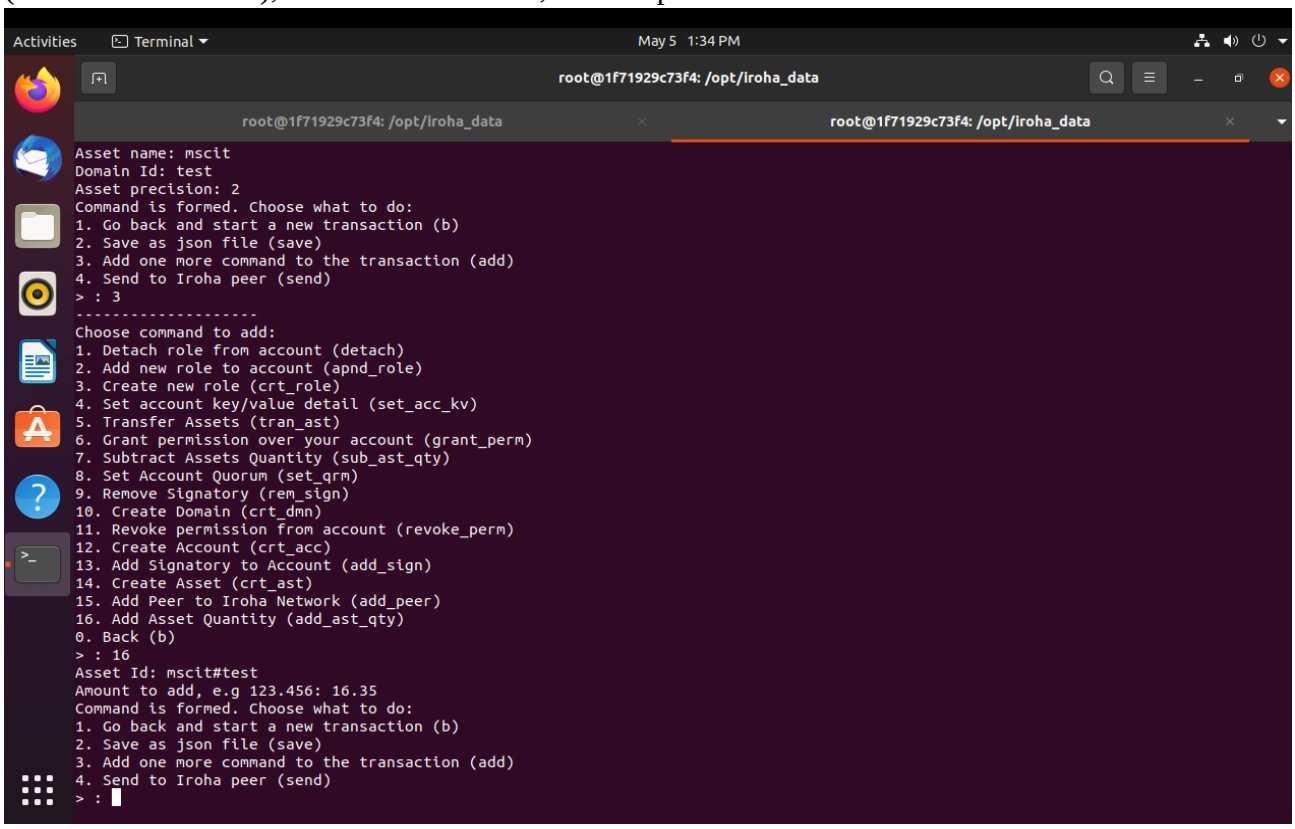
Step 9: We should be inside the docker container's shell. Launch the iroha-cli tool and login *iroha-cli -account_name admin@tests admin@test.*



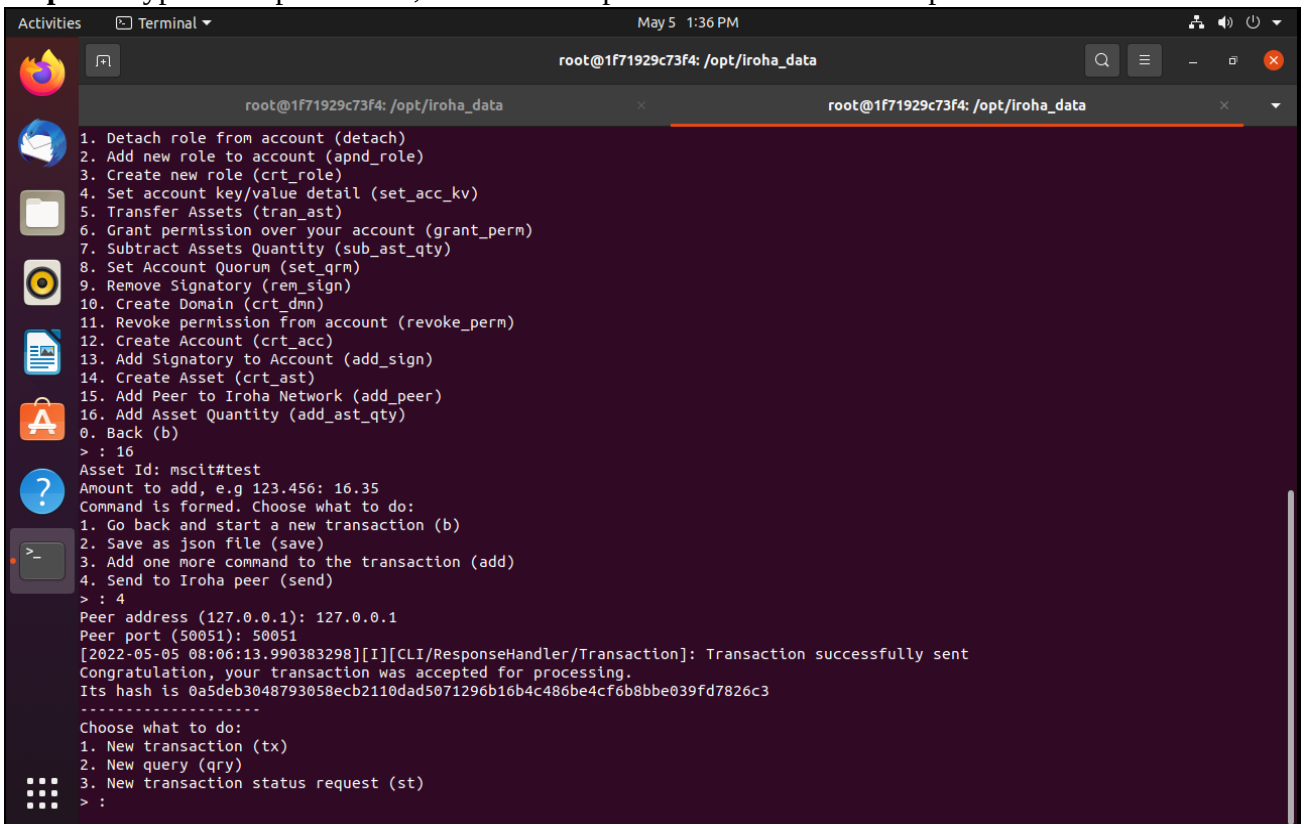
Step 10: Type 1 and press enter to start a new transaction. Then Type 14 and press enter to create asset. Enter Asset name: mscit Domain Name: test Asset precision: 2



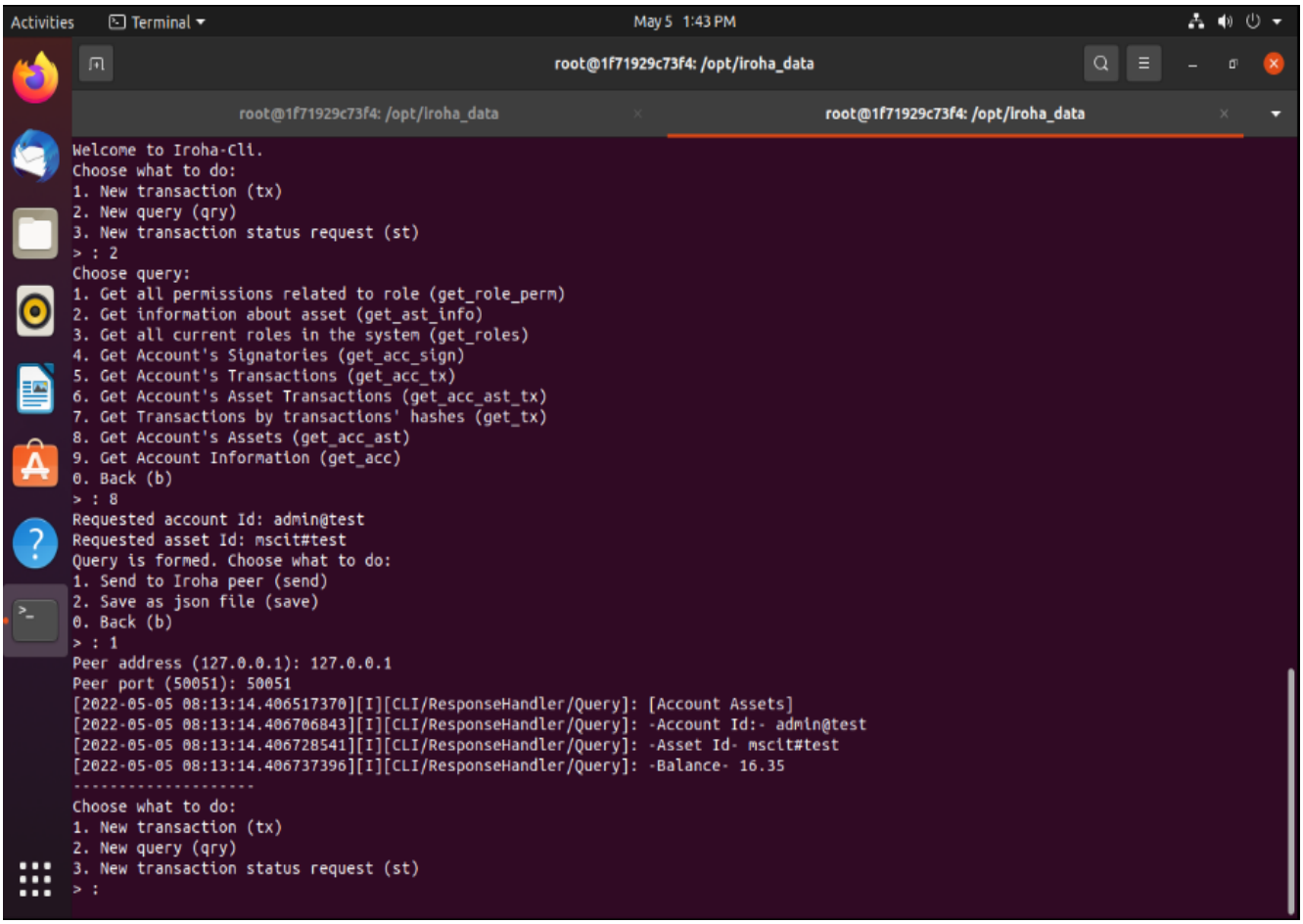
Step 11: Type 3 and press enter to add one more command to the transaction. Then Type 16 and press enter to add Asset Quantity. The account is "admin@test", the asset is "mscit#test" (notice the #domain), the amount is 12.56, and the precision is 2.



Step 12: Type 4 and press enter; send it to the peer address 127.0.0.1 at port 50051.



Step 13: Type 2 and press enter to make a new query this time. We're going to query a particular account's assets, so type 8 and press enter. The account that owns the amount is "admin@test", the asset we need is "mscit#test". Send the query request to "127.0.0.1" at port "50051"

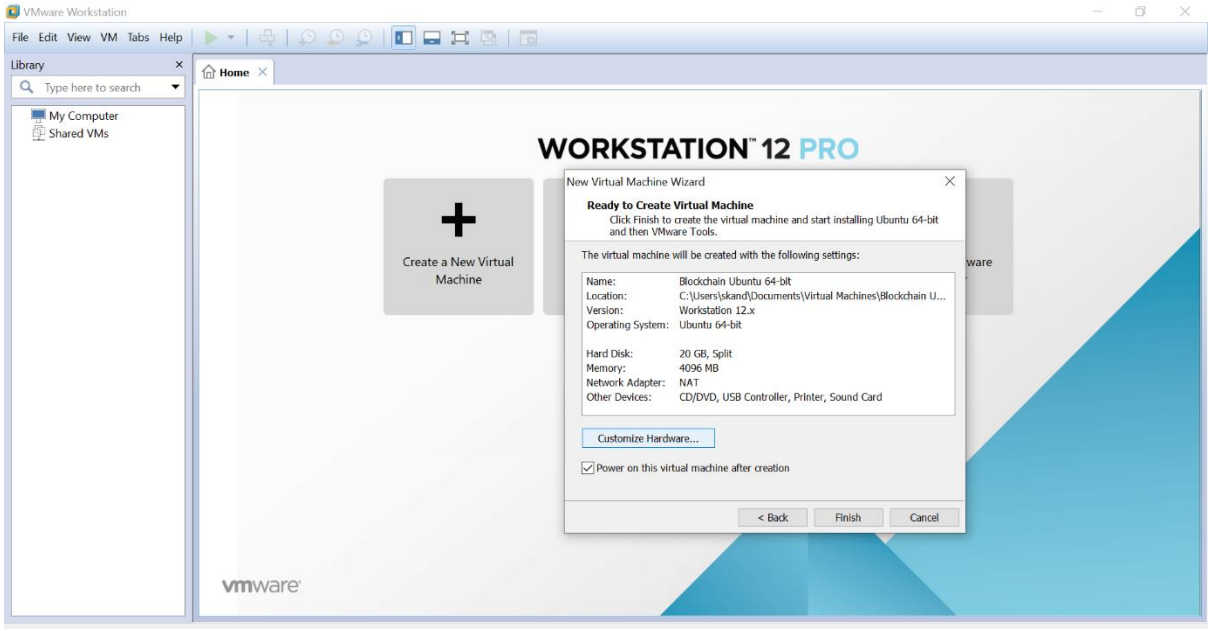


PRACTICAL NO. 6

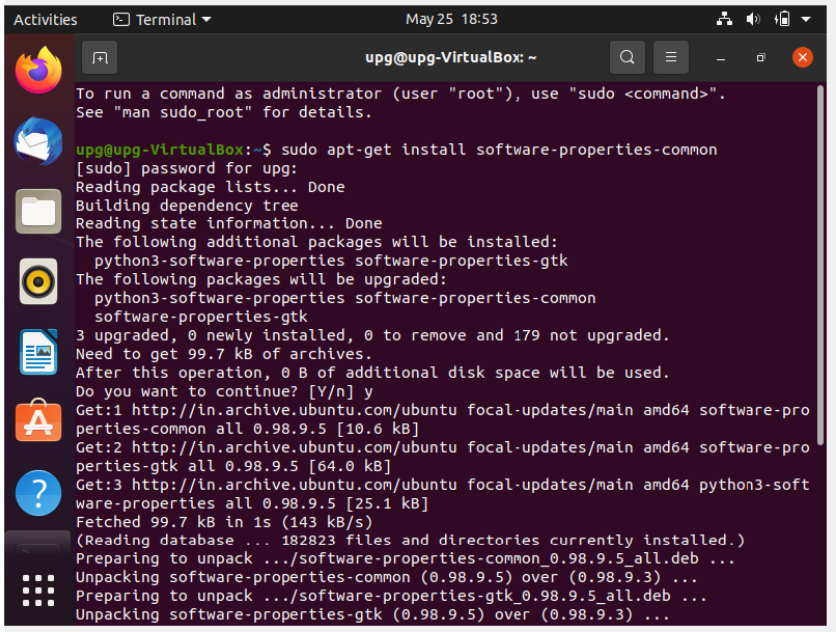
Aim : Write a program to demonstrate mining of Ether.

Steps :

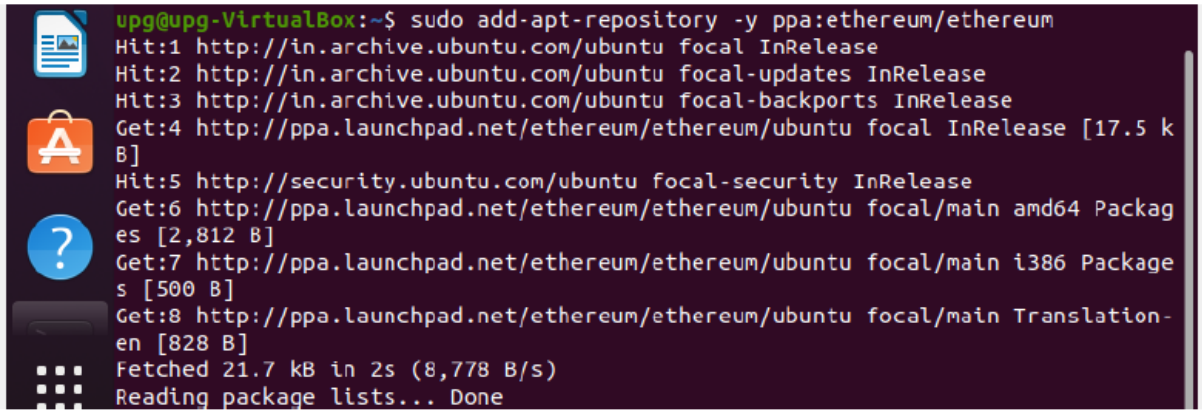
Step 1: Firstly, we need to install Virtual Box. Once done with the installation process, create the Virtual Machine. After creating Virtual Machine, your machine will be created but some settings are required. Go to the Settings option as you can see below.



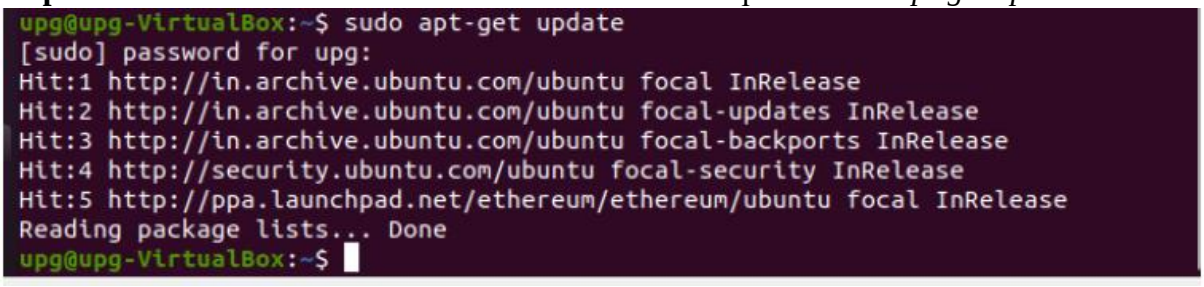
Step 2: Select the Virtual Machine, here we are going to select practical 6 machine and click on start. Once the machine is on you need to go the search bar and type terminal. After opening the terminal, you need to type the following command:
sudo apt-get install software-properties-common.



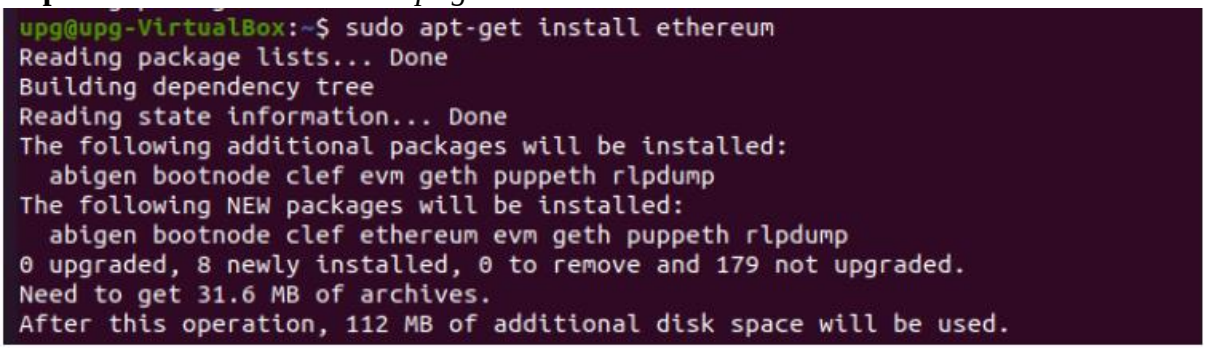
Step 3: Type the second command as shown below:
sudo add-apt-repository -y ppa:ethereum/Ethereum



Step 4: After done with the above command it's time to update. *sudo apt-get update*



Step 5: Install Ethereum. *sudo apt-get install ethereum*



Step 6: On Completing the installation process of Ethereum you need to run Geth. So here you need to type Geth in the terminal.

Step 7: To restart Geth with the console, type the following command:
geth
geth attach

Step 8: So your Geth Client should be running with the concole enabled, giving you the command prompt. We will be creating an account by using the Javascript API call, but you need to keep in mind that the geth synchronization needs to be completely synchronized. So, in Console type: personal.newAccount("*your_new_account_password_here*")
Note, replace the text between the quotes with the password.

Step 9: After creating an account you will be returned with the public key which will be displayed in green color. By typing: *personal.listAccounts*
It is possible for you to see all your accounts in the console.

Step 10: You can do all the above things in other way that is by using flags. Here we can use Geth Command line to launch Geth with certain flags. In order to get started with this start Geth on testnet by typing following command: *geth - -testnet*

Step 11: Geth will not begin mining automatically you will have to command it to start or stop mining. In order to begin with mining on the main network, open a new terminal and enter JavaScript console by typing the following command: *geth*

Step 12: You need to tell your node the Ethereum address for receiving your mining payments. This can be done by typing the following command in the console:
mine.setTherbase(eth.accounts[*your_address_here*])
Remember to replace quotes with your address.

Step 13: Start the Mining Process using the command: *miner.start()*
You can stop this process by typing: *miner.stop()*

PRACTICAL NO. 7

Aim: Demonstrate the running of the blockchain node.

Theory :

The main purpose behind having a Blockchain application is to ensure the integrity of data and provide credibility to the network. These properties are maintained by sharing the same ledger amongst different systems, distributed globally.

The concept of Blockchain states that every block that contains data is connected cryptographically to the next block. If you change or modify any block, all the subsequent blocks have to be changed as their value or their hash depends on the previous block. However, if this cryptographic chain of blocks was present at a single location, any hacker could change the value of all the blocks. In order to sustain the integrity of data, the Blockchain is a distributed network that shares the same copy of ledger or data among multiple systems.

This is the reason why every Blockchain network requires globally distributed systems to be truly decentralized.

If the ledger of any one system is changed, the ledger on the other systems acts as proof of the integrity of data. Therefore, the availability of these systems or Blockchain nodes is what allows it to be distributed and trustworthy. Without nodes, Blockchain is just a database secured with cryptographic hashing!

Regardless of the Blockchain being public or private blockchain, a globally distributed network is essential for the perseverance of the data, achieved by saving the transactional record on the Blockchain nodes. The immutable record on nodes is the reason behind the unquestionable auditability of data or transaction records on Blockchain.

Moreover, having different Blockchain nodes across the global network allows the network to be resilient towards any centralized attacks or even from any natural calamities.

Even if a whole nation gets destroyed due to any reason, all it needs is one Blockchain node to provide the Blockchain ledger to the network.

PRACTICAL NO. 8

Aim : Demonstrate the use of Bitcoin Core API.

Code :

```
# -*- coding: utf-8 -*-
"""
Created on Mon May  2 15:00:32 2022
@author: Sahil Kandalkar
"""
# Sahil Kandalkar - 53004200009
from bitcoinlib.wallets import Wallet

w = Wallet.create('Wallet2')
key1 = w.get_key()
print(key1.address)
w.scan()
print(w.info())
```

Output :

```
Console 1/A

In [4]: runfile('S:/Sahil Kandalkar/M.Sc IT/Sem 4 Practicals/BLOCKCHAIN/BLK_Prac8_BitcoinCore_API/BLK_Prac8_BitcoinCore_API.py', wdir='S:/Sahi
Practicals/BLOCKCHAIN/BLK_Prac8_BitcoinCore_API')
19BFRqQfCFSVDMcZU6fVn4SGo4HYxeRYNV
=== WALLET ===
 ID                          2
 Name                        Wallet2
 Owner
 Scheme                      bip32
 Multisig                    False
 Witness type                legacy
 Main network                bitcoin
 Latest update               2022-05-14 23:41:45.507654

= Wallet Master Key =
 ID                          17
 Private                     True
 Depth                       0

- NETWORK: bitcoin -
- - Keys
   22 m/44'/0'/0'/0/0            19BFRqQfCFSVDMcZU6fVn4SGo4HYxeRYNV          address index 0                0.00000000 ฿
   23 m/44'/0'/0'/0/1            1H6zxWFgu27ALzhqFoV7D31hst8gkywp5K          address index 1                0.00000000 ฿
   24 m/44'/0'/0'/0/2            1GJa1zbK3yre9qsLbsXhJt6Ge5eogvcrRo          address index 2                0.00000000 ฿
   25 m/44'/0'/0'/0/3            1AaMwZAMy9gpV5Axt4BzzDWzFtMs8wbrAy          address index 3                0.00000000 ฿
   26 m/44'/0'/0'/0/4            1G45kJUH9MubvxjoiJinmewUK1fo6YCi1k          address index 4                0.00000000 ฿
   28 m/44'/0'/0'/1/0            144hX8S2BTqyJHfspQbggeRzVsnY9VCKH4          address index 0                0.00000000 ฿
   29 m/44'/0'/0'/1/1            1HPD5pAV1sRgS4RpNaM5jxxkhXFE5qiVTi          address index 1                0.00000000 ฿
   30 m/44'/0'/0'/1/2            115K136RXPtowFtXRsE6magSZBzBXEwjpM          address index 2                0.00000000 ฿
   31 m/44'/0'/0'/1/3            1NcvMZvdxGgq5JS9hLrCoPTSCcCbPLsGJS          address index 3                0.00000000 ฿
   32 m/44'/0'/0'/1/4            1BCfwFSMojStb4LC4PKLuJXSSUuwHeq1iv          address index 4                0.00000000 ฿

- - Transactions Account 0 (0)

= Balance Totals (includes unconfirmed) =


None

In [5]:
```

PRACTICAL NO. 9

Aim: Create your own blockchain and demonstrate its use.

Steps :

Step 1: To enable our launchpad repository run:

- *sudo add-apt-repository -y ppa:ethereum/Ethereum*

```
mscit@ubuntu:~$ sudo add-apt-repository -y ppa:ethereum/ethereum
gpg: keyring `/tmp/tmpn1mpbc15/secring.gpg' created
gpg: keyring `/tmp/tmpn1mpbc15/pubring.gpg' created
gpg: requesting key 923F6CA9 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmpn1mpbc15/trustdb.gpg: trustdb created
gpg: key 923F6CA9: public key "Launchpad PPA for Ethereum" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
OK
mscit@ubuntu:~$
```

- Then install the stable version of go-ethereum: *sudo apt-get update*

```
mscit@ubuntu: ~
mscit@ubuntu:~$ sudo apt-get update
Hit:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
Get:2 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial InRelease [15
B]
Hit:3 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:5 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease
Get:6 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial/main amd64 Pa
es [2,764 B]
Get:7 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial/main i386 Pacl
s [2,800 B]
Get:8 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial/main Translat
en [828 B]
Fetched 21.8 kB in 1s (13.8 kB/s)
Reading package lists... Done
mscit@ubuntu:~$
```

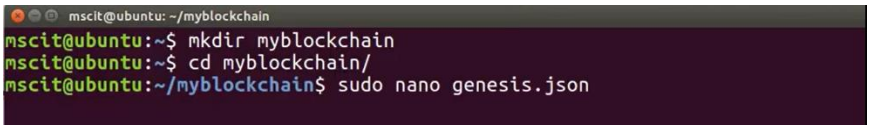
- *sudo apt-get install ethereum*

```
mscit@ubuntu: ~
mscit@ubuntu:~$ sudo apt-get install ethereum
[sudo] password for mscit:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  abigen bootnode clef evm geth puppeth rlpdump
The following NEW packages will be installed:
  abigen bootnode clef ethereum evm geth puppeth rlpdump
0 upgraded, 8 newly installed, 0 to remove and 432 not upgraded.
Need to get 30.4 MB of archives.
After this operation, 104 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

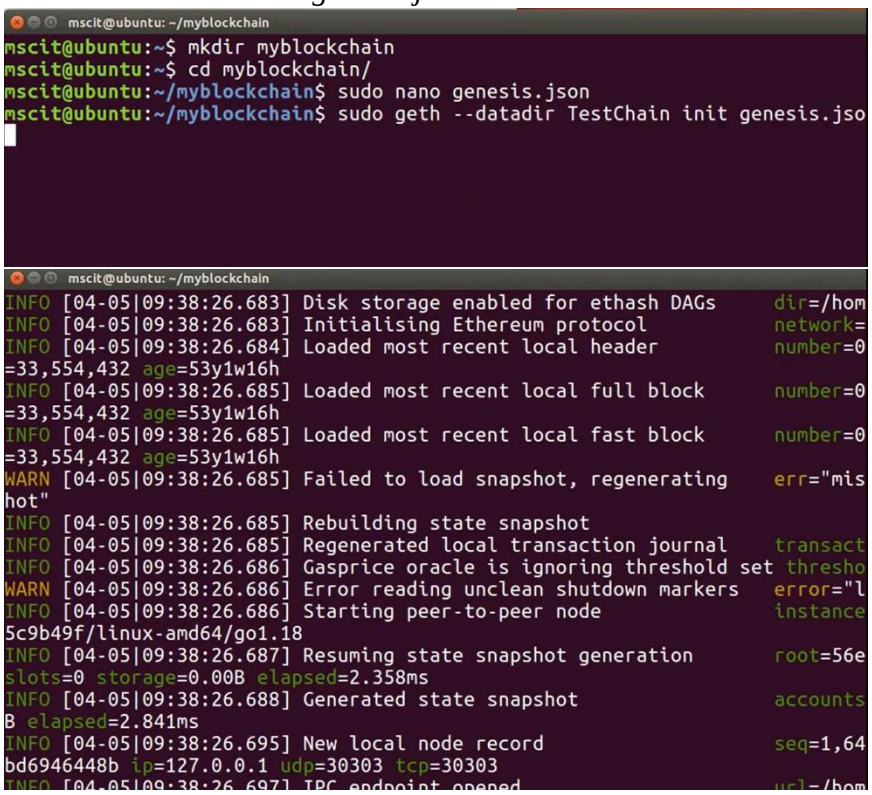
Step 2: Create a new directory: *mkdir myblockchain.*
Navigate to myblockchain: *cd myblockchain/*

Create a genesis.json file:
sudo nano genesis.json
Copy and paste the below code, ctrl+o (write), press enter, ctrl+x (exit).

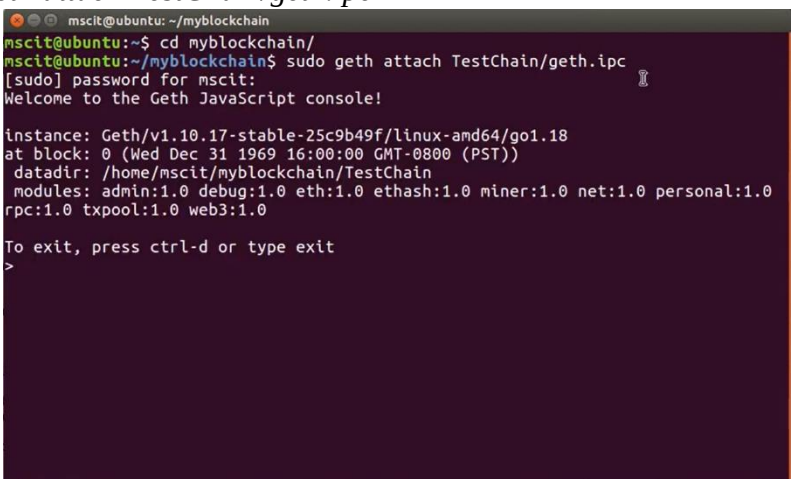
```
{
"config":{
"chainId": 10,"homesteadBlock": 0,"eip155Block": 0,"eip158Block": 0,"eip150Block": 0,
"eip150Hash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
},
"alloc": {},
"coinbase": "0x0000000000000000000000000000000000000000",
"difficulty": "0x02000000",
"extraData": "",
"gasLimit": "0x2fefd8", "nonce": "0x0000000000000042",
"mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
"parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
"timestamp": "0x00"
}
```



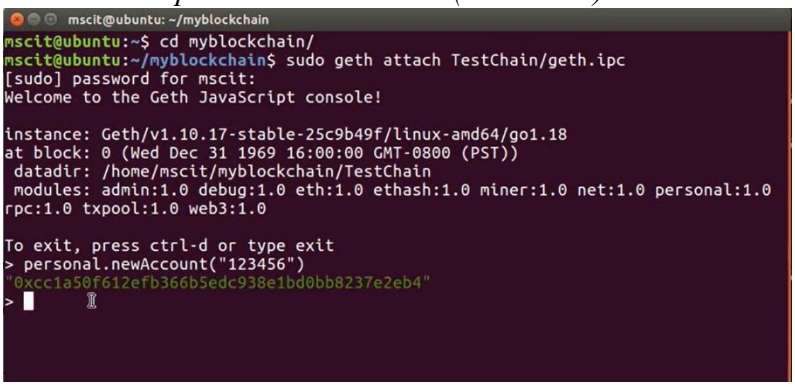
Step 3: Initialize the genesis block:
sudo geth --datadir TestChain init genesis.json



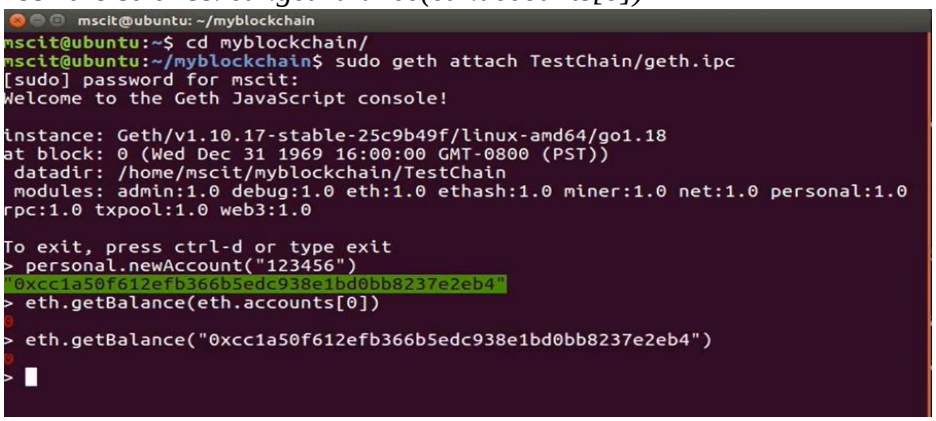
Step 4: Open a new terminal. Navigate to myblockchain and Run the below line to open geth console: *sudo geth attach TestChain/geth.ipc*



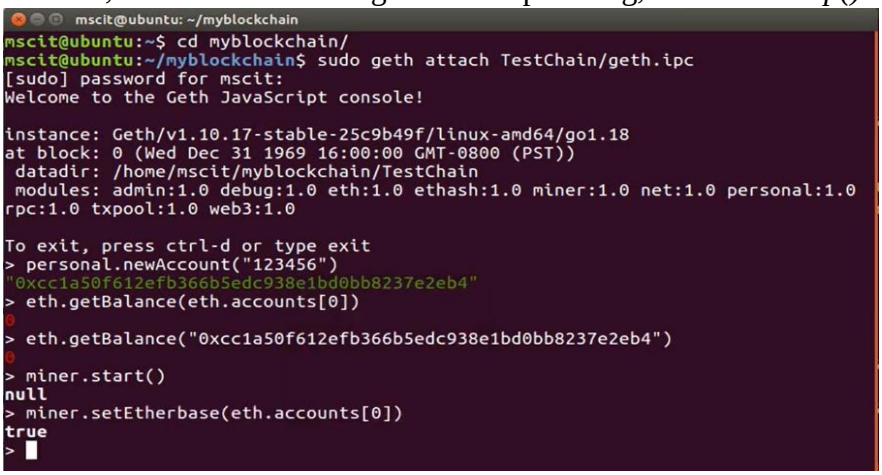
Step 5: Create new account: *personal.newAccount("123456")*



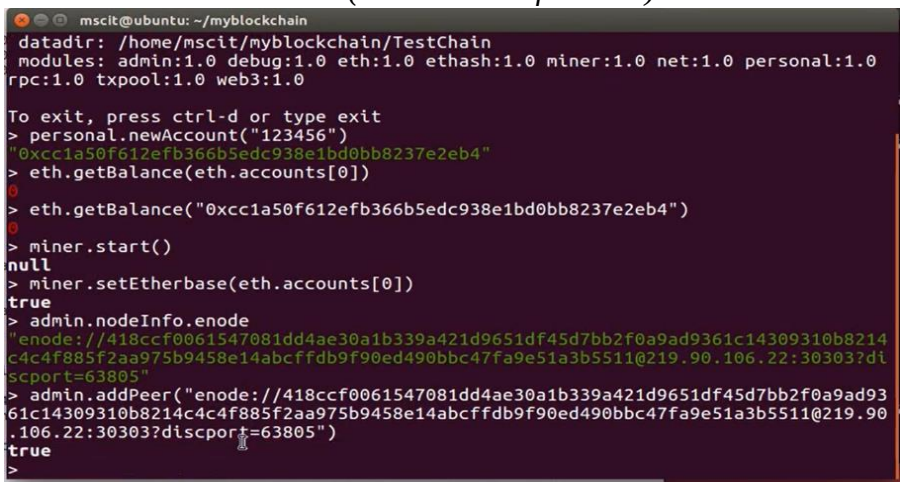
Step 6: Check the balance: *eth.getBalance(eth.accounts[0])*



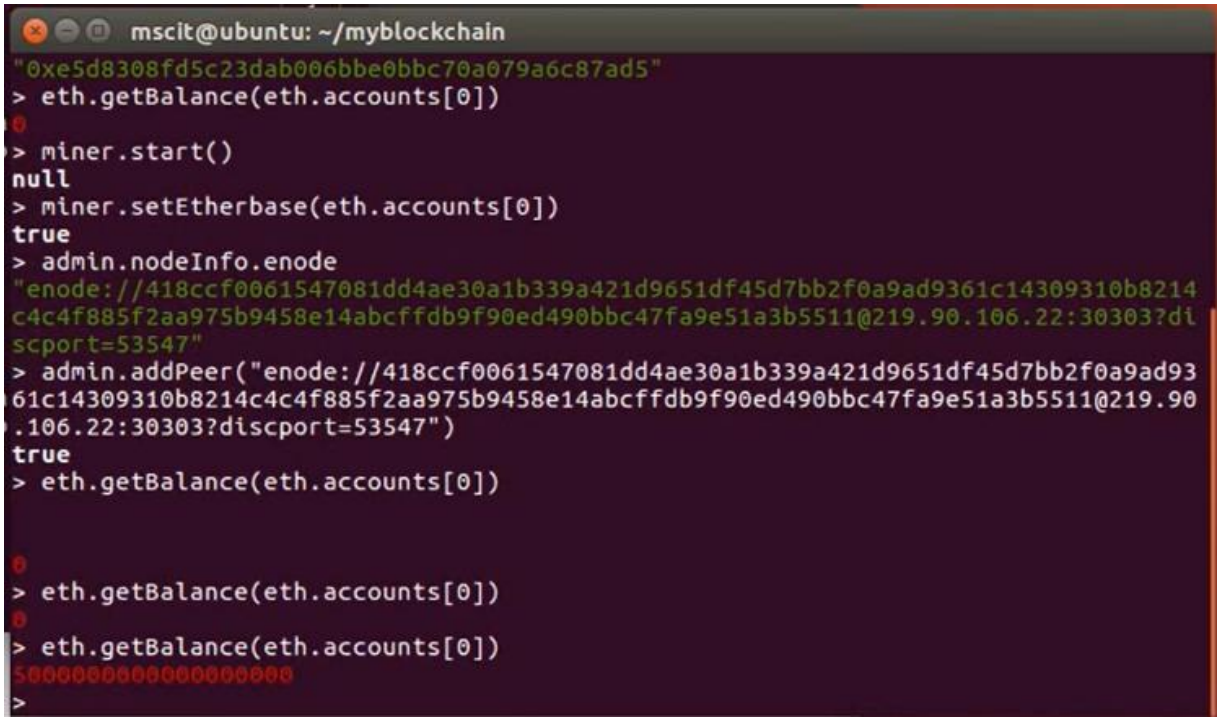
Step 7: There are two ways to get ether to your account. You can mine blocks and get rewarded with ether or someone sends you some ether to your account. Since you are alone in your private network at this point, the only way is to mine some blocks and get rewarded.
Mining in the main Ethereum network is hard and need more computational power. However, mining blocks in a private network is easy since we specified a low difficulty level in the genesis.json file. To mine ether to your account, simply type *miner.start()* in the Geth console. After 10–15 seconds, check the balance again. To stop mining, run *miner.stop()*



Step 8: To add nodes: *admin.addPeer(admin.nodeInfo.enode)*



Output:


mscit@ubuntu: ~/myblockchain
"0xe5d8308fd5c23dab006bbe0bbc70a079a6c87ad5"
> eth.getBalance(eth.accounts[0])
0
> miner.start()
null
> miner.setEtherbase(eth.accounts[0])
true
> admin.nodeInfo.enode
"enode://418ccf0061547081dd4ae30a1b339a421d9651df45d7bb2f0a9ad9361c14309310b8214c4c4f885f2aa975b9458e14abcffdb9f90ed490bbc47fa9e51a3b5511@219.90.106.22:30303?discport=53547"
> admin.addPeer("enode://418ccf0061547081dd4ae30a1b339a421d9651df45d7bb2f0a9ad9361c14309310b8214c4c4f885f2aa975b9458e14abcffdb9f90ed490bbc47fa9e51a3b5511@219.90.106.22:30303?discport=53547")
true
> eth.getBalance(eth.accounts[0])
0
> eth.getBalance(eth.accounts[0])
0
> eth.getBalance(eth.accounts[0])
5000000000000000000
>

PRACTICAL NO. 10

Aim: Build Dapps with angular.

Theory :

Angular is a TypeScript-based open-source framework, whose main purpose is developing web applications. But where Angular really shines is the creation of client applications and it is regarded as one of the best tools when it comes to single-page apps development as well Angular community consists of more than 18 million users and that number is simply impressive.

A big part in this is played by tools and setup that Angular possesses: RxJS, Angular CLI, and different code editors that support the framework.RxJS is a reactive programming library that is pretty crucial when it comes to working with Angular. RxJS' main aim is to handle asynchronous data with multiple events and by doing that this reactive programming library allows engineers to build multiple channels of data exchange to ease the consumption of resources. So, basically, RxJS is similar to a conveyor for JavaScript codes, as it allows parallel and continuing execution of events in a manner, independent from one another, and without waiting for one event to happen to complete another. And although RxJS' learning curve might seem a bit high, it is worth every penny.

The second great tool in Angular's arsenal is the Angular command-line interface (or just CLI for short). Favored by many an engineer, Angular CLI is easy to set up, quite understandable even for newcomers, is packed to the brim with different testing tools right out of the box and its commands can be described as nothing but simple.

Angular Pros

1.Angular architecture is component-based and its primary architectural characteristic is the basis of components hierarchy. This fact allows developers to achieve a higher code quality by making the overall code more accessible and understandable by encapsulating all of the components with their functionality.

2.All Angular components are reusable. And this advantage is a direct outcome of the previous one because the previously mentioned encapsulation of components makes them exceptionally self-sufficient. It also allows developers to reuse them in different parts of their applications, making the process of developing a bit faster and more convenient.

3.Angular's readability is off the charts. Once again due to the component-based architecture and the encapsulation. Thus, new developers, albeit new to the whole

app developing a game or just new to the project, can read code in a better way and reach their plateau of productivity quicker.

Angular is unit-test friendly. Try and guess why it is so. Right you are, all because of the component-based structure that simplifies the quality assurance procedures even when it comes to the smallest parts of the app, which are, of course, units.

Angular uses TypeScript. Let us get a little misconception out of the way first – it is not mandatory to use TypeScript with Angular, as it provides devs with options on how to use their libraries more efficiently, including Redux and Flux. But why use them if you can use TypeScript, which can be described as a superset for JavaScript? Yes, it has its fair share of things to nitpick and yes, you basically have to learn another language if you never worked with TypeScript, but its overall usefulness is immense. Especially if you work on an enterprise-level project, as TypeScript simply has better navigation, autocompletion, refactoring services and it helps you to spot and get rid of common mistakes while you type in the code. All in all, TypeScript is great, and Angular is only better because of it.

Angular Cons

The main disadvantage of Angular is the fact that Angular is complex. Although the component-based architecture Angular possesses is great, the way in which components are managed is not, as each and every component in your app will, most likely, need dependency injections and all of them will definitely need lifecycle interfaces. And that's not mentioning the fact that you will have to rely on third-party libraries that are quite specific when it comes to Angular. Thus, developing apps on Angular can be (bear in mind that it is a possibility and not an axiom) pretty repetitive and tiresome.

Summing up, even though it can be fearsome for new developers and at times complicated for new and professional developers alike, Angular is a great tool for a variety of web app development needs and, with Google's Long-Term Support, its future in the industry looks as bright as ever. That being said, let's get to the metaphorical cherry on top of the cake of today's article – an example of step-by-step Angular app development.

Step-by-step Angular App Creation

1. Install Angular CLI 8;
2. Proceed with Angular 8 Project creation;
3. Add Angular HttpClient;
4. Create UI Component;
5. Routing addition;
6. Build UI with Angular Material Components;
7. Set up a REST API mocking;
8. Use Angular HttpClient in order to consume the REST API;
9. HTTP Errors Handling is step number nine;
10. Pagination Addition;
11. Angular Application Firebase building and deployment.