PRACTICAL REPORT

ON

PPSCSP3011: WEB 3 TECHNOLOGIES

SUBMITTED BY

PANKAJ LILADHAR PATHAK

ROLL NO: 29

SUBMITTED TO

Ms. NAMRATA KAWALE

MSc. (COMPUTER SCIENCE) SEM - III

2022 – 2023

CONDUCTED AT

CHIKITSAK SAMUHA'S
S. S. & L.S. PATKAR COLLEGE OF ARTS & SCIENCE
AND
V. P. VARDE COLLEGE OF COMMERCE & ECONOMICS
An Autonomous college,
Affiliated to University of Mumbai
GOREGAON (W). MUMBAI -400062

CHIKITSAK SAMUHA'S

SIR SITARAM & LADY SHANTABAI PATKAR COLLEGE OF ARTS & SCIENCE

&

V.P. VARDE COLLEGE OF COMMERCE & ECONOMICS

GOREGAON (WEST), MUMBAI - 400 104.

An Autonomous College, University of Mumbai

CERTIFICATE

Certified that such of the experiments as have been duly signed were performed by Mr./Miss _____

Roll No. _____ *of* _____ *class* _____

Division _____ *in the* _____ *Laboratory of this college during the year* _____

Professor-in-Charge **Examiner** **Co-ordinator**

Date: _____ _____ *Department*

Practical No	Practical Aim	Date	Sign
1	Install and understand Docker container, Node.js, Ethereum and perform necessary software installation on local machine.	14/09/2022	
2	A Simple Client Class That Generates the Private and Public Keys by Using the Built in Python RSA Algorithm and Test.	26/09/2022	
3	A Transaction Class to Send and Receive Money and Test It.	26/09/2022	
4	Create Multiple Transactions and Display Them.	26/09/2022	
5	Create A Blockchain, A Genesis Block and Execute It.	30/09/2022	
6	Create A Mining Function and Test It.	26/09/2022	
7	Add Blocks to The Miner and Dump the Blockchain.	26/09/2022	
8	Implement and Demonstrate the Use of Solidity Programming: A. Your First Solidity Smart Contract (Counter Program). B. To Create and Explore Types of Variables with Varying Data Types in Solidity Programming (Variables).	26/09/2022	
9	Develop a decentralized voting application using solidity.	30/09/2022	

Practical No: 1

Aim: Install and understand Docker container, Node.js, Ethereum and perform necessary software installation on local machineWhat is docker?

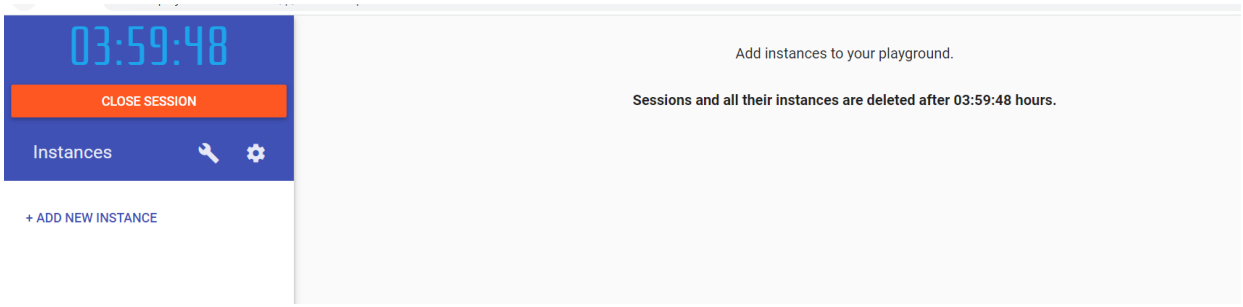
- **Docker:** Docker is a containerization platform that packages your application and all its dependencies together in the form of a docker container to ensure that your application works seamlessly in any environment.
- **Container:** Docker Container is a standardized unit which can be created on the fly to deploy a particular application or environment. It could be an Ubuntu container, CentOs container, etc. to fullfill the requirement from an operating system point of view. Also, it could be an application oriented container like CakePHP container or a Tomcat-Ubuntu container etc.
- **Node.js:** Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on a JavaScript Engine (i.e. V8 engine) and executes JavaScript code outside a web browser, which was designed to build scalable network applications. Node.js lets developers use JavaScript to write command line tools and for server-side scripting— running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.
- **Ethereum:** Ethereum is a decentralized, open-source blockchain with smart contract functionality. Ether is the native cryptocurrency of the platform. Among cryptocurrencies, ether is second only to bitcoin in market capitalization. Ethereum was conceived in 2013 by programmer Vitalik Buterin.
- **Remix IDE:** Remix IDE is an open source web and desktop application. It fosters a fast development cycle and has a rich set of plugins with intuitive GUIs. Remix is used for the entire journey of contract development with Solidity language as well as a playground for learning and teaching Ethereum.https://labs.play-with-docker.com/

Steps:

Step0:-

A. Docker container (Web Based)

1. Open your browser and go to https://labs.play-with-docker.com/, then play with docker page will open up here click on login button and select docker option.
2. Now new sign up page will occur on new window, here click on “Sign Up” option from top right corner and fill up the details to create a docker account and click sign up button to proceed.
3. After the sign-up process you get back to a main Log In page. Now just fill your Login Credentials and click on continue now select the $0 personal plan on next page to continue free.
4. Then click on Start button, after you click on start button your free docker session gets started.



5. Now in docker playground for the web based docker container click on ADD NEW INSTANCE and get the instance as shown below

.

Step1:- $docker –version

$docker pull rocker/verse or busybox

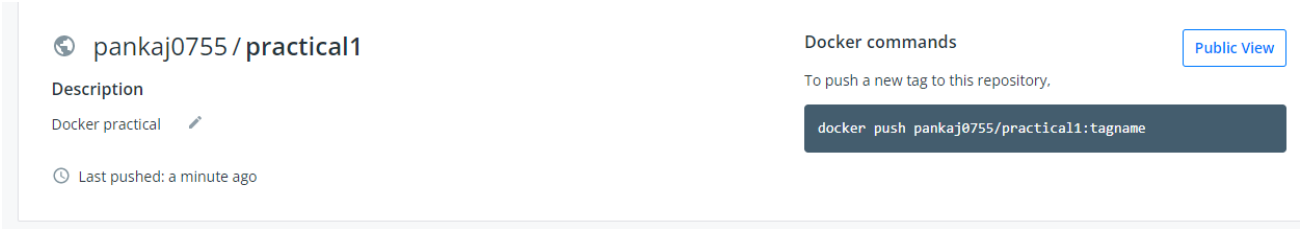
```
# The PWD team.                                                #
################################################################
[node1] (local) root@192.168.0.18 ~
$ docker --version
Docker version 20.10.17, build 100c701
[node1] (local) root@192.168.0.18 ~
$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
729ce43e2c91: Pull complete
Digest: sha256:135c42550cbea37749b2040355ea1d58245334920ce88706ac6d2adb2d84d3fd
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
```

Step2:- $docker images - check image file

```
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
busybox      latest    2bd29714875d   6 hours ago    1.24MB
[node1] (local) root@192.168.0.18 ~
```

Step3:- go to the link and Login In with docker account, then in docker hub page click on “Create a Repository” option. https://hub.docker.com/

- Now give repository a name, small description and visibility as Public. Your repository will look like given below in second image after creation.



Step4 :- docker login --username=(your username) also give the password (shows invisible)> to connect with docker hub repository.

docker login --username=pankaj0755

```
[node1] (local) root@192.168.0.18 ~
$ docker login --username=pankaj0755
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[node1] (local) root@192.168.0.18 ~
```

Step5:- docker tag (IMAGE ID) (Username/repo name):t1 > for tagging the repository

$docker tag image_Id/pankaj0755/practical1:tagname

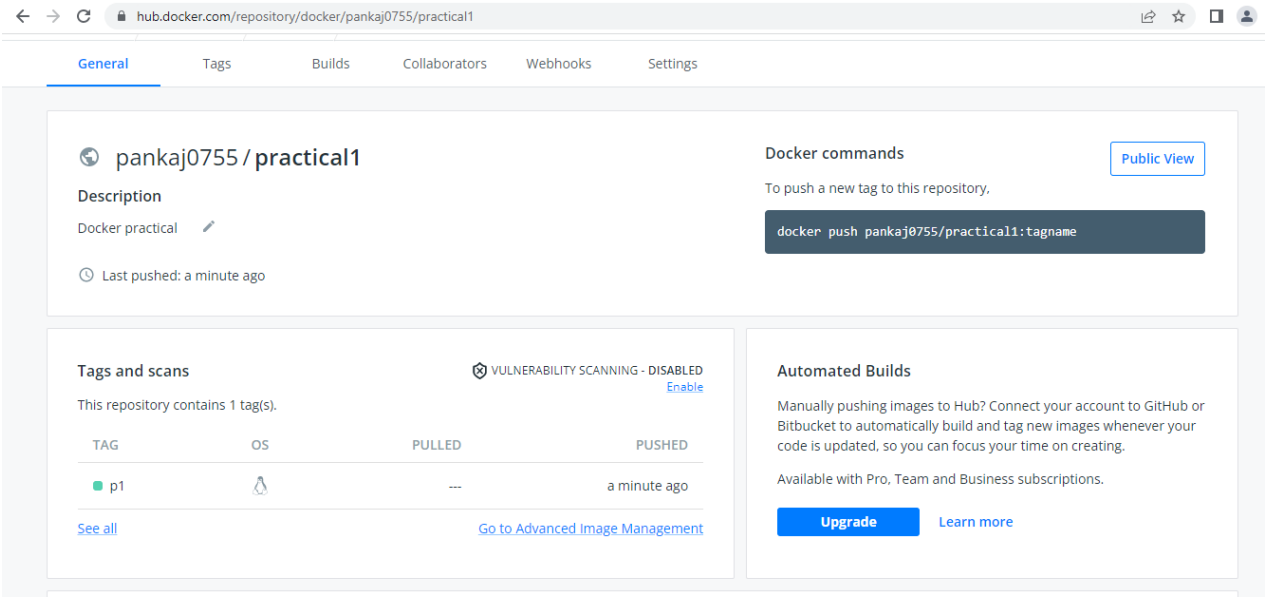
```
[node1] (local) root@192.168.0.18 ~
$ docker tag 2bd29714875d pankaj0755/practical1:p1
[node1] (local) root@192.168.0.18 ~
```

Step 6:- docker push (Username/repo name):t1 > pushing tag to repository

$docker push pankaj0755/practical1:tagname

```
[node1] (local) root@192.168.0.18 ~
$ docker push pankaj0755/practical1:p1
The push refers to repository [docker.io/pankaj0755/practical1]
3e9498aeb76f: Mounted from library/busybox
p1: digest: sha256:bbb248c803ff97f51db3b37a2a604a6270cd2ee1ca9266120aeccb3b19ce80d2 size: 527
[node1] (local) root@192.168.0.18 ~
$
```

Step7:- After the successful tagging, in Tags and scans section of docker hub you can see the tagged



B. Node.js

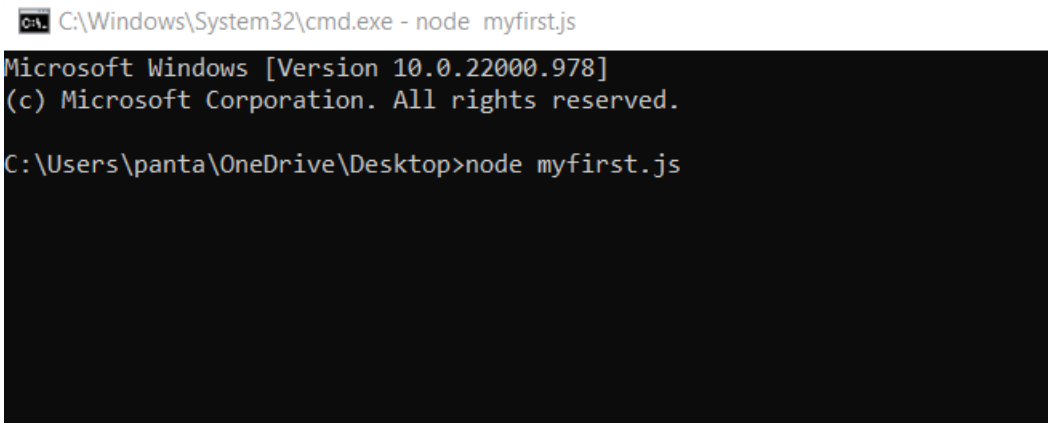
- ❖ Install the stable recommend Node.js version

Inbuild module Program

1. For that first create the notepad file and add the below code and save it as myfirst.js:

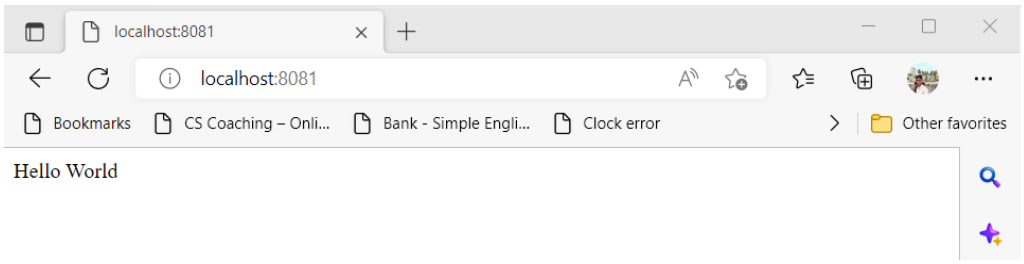
```
var http = require("http");

http.createServer(function(req,res)

{

   res.writeHead(200,{'Content-Type':'text/html'});

   res.end('Hello World');

}).listen(8081);
```

2. Now go to the file location where the file is created and open the command prompt or windows power shell and type node myfirst.js, if everything is correct you can see the command prompt as shown below



3. Now open your browser and type localhost (and port number which you given in code). Note: if 8081 port not working on your machine try changing it to 8082 or 8080

Output:-



User Define Code

1. First create the module that returns the date using below code (save as myfirstmodule.js):

```
exports.myDateTime = function()

{

    return Date();

};
```

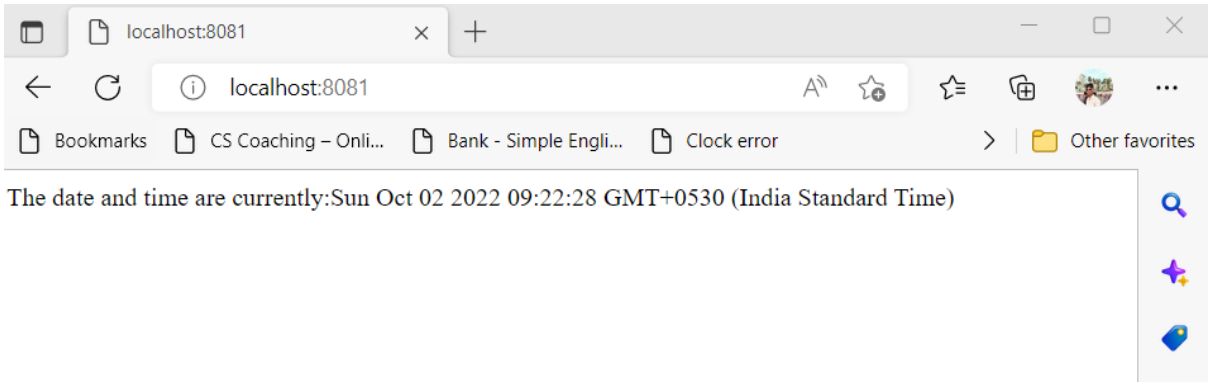
2. For main program code (save as main.js):

```
var http = require('http');
var dt = require('./myfirstmodule');
http.createServer(function (req, res){
 res.writeHead (200,{'Content-Type':'text/html'});
 res.write("The date and time are currently:" +dt.myDateTime());
 res.end();
}).listen(8081);
```

3. Now go to the file location where the file is created and open the command prompt or windows power shell and type node main.js, if everything is correct you can see the command prompt as shown below.

4. Now open your browser and type localhost (and port number which you given in code). Note: if 8081 port not working on your machine try changing it to 8082 or 8080

Output:

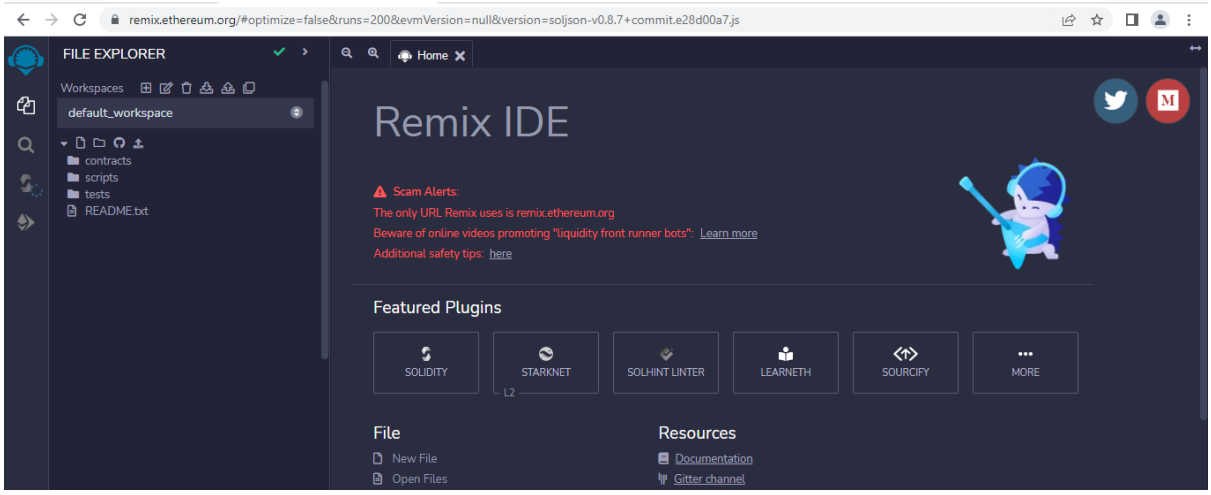


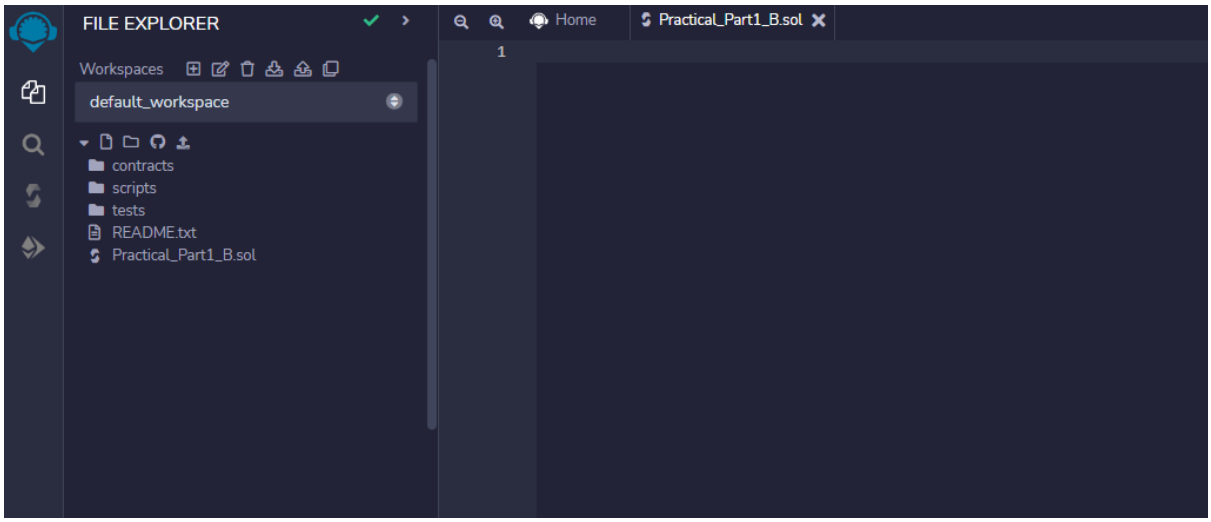
C. Ethereum (Remix IDE)

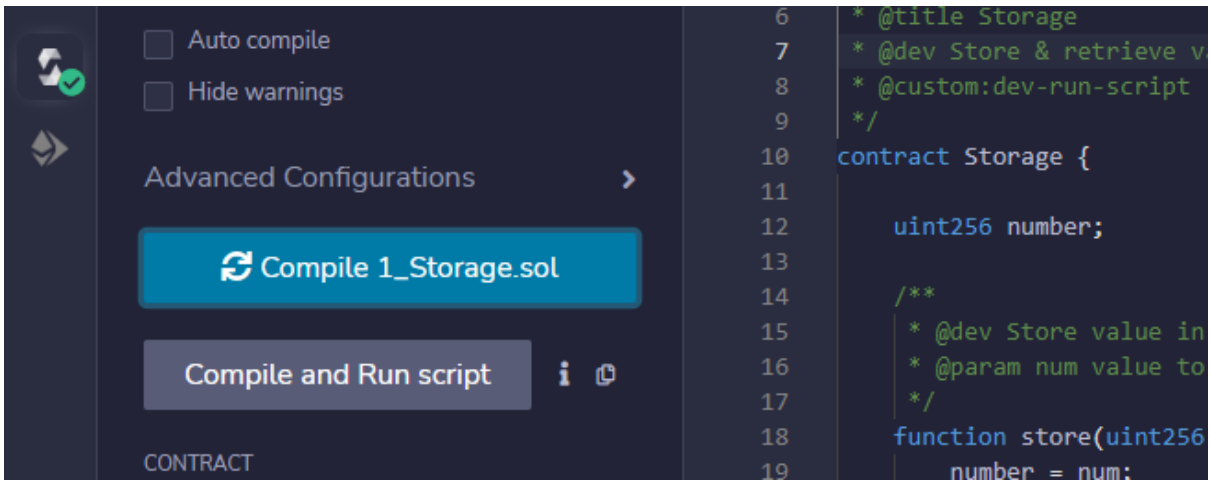
1. Go to https://remix.ethereum.org/

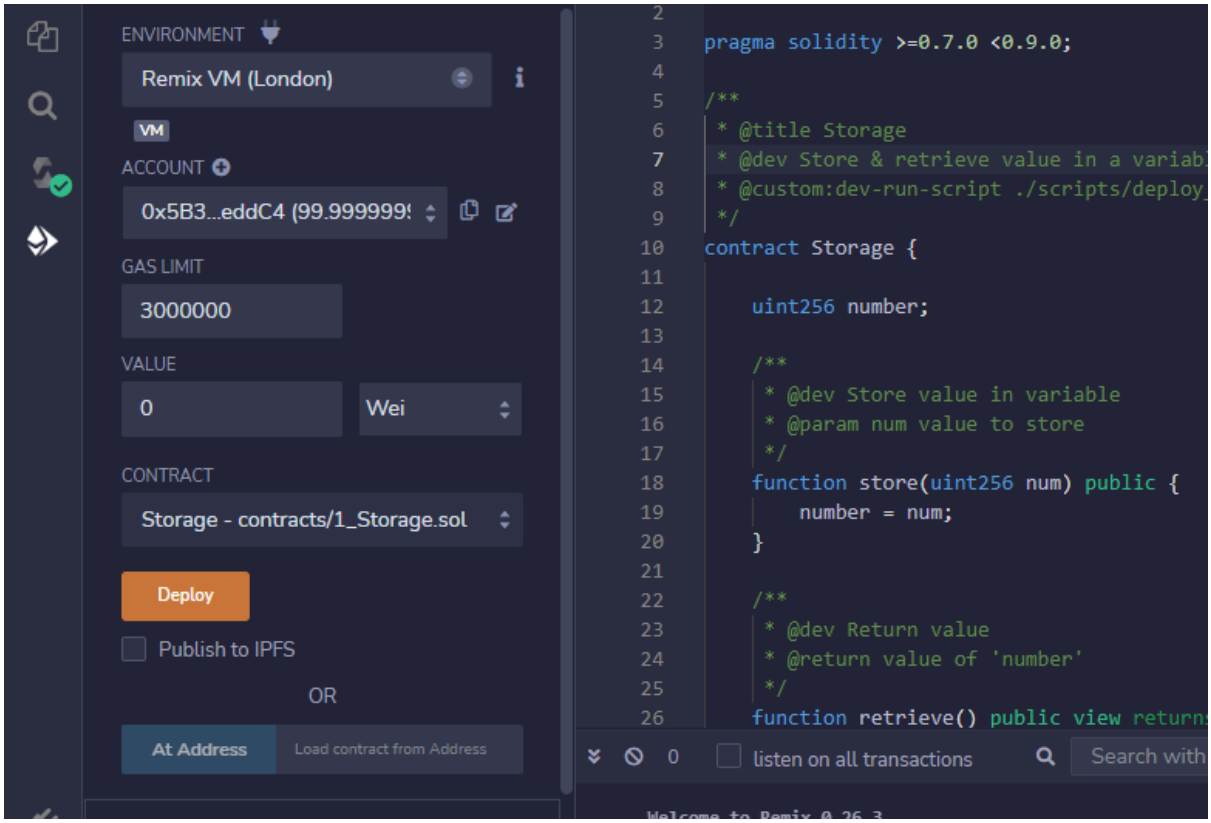
2. You get treated with Remix IDE for Ethereum coding.

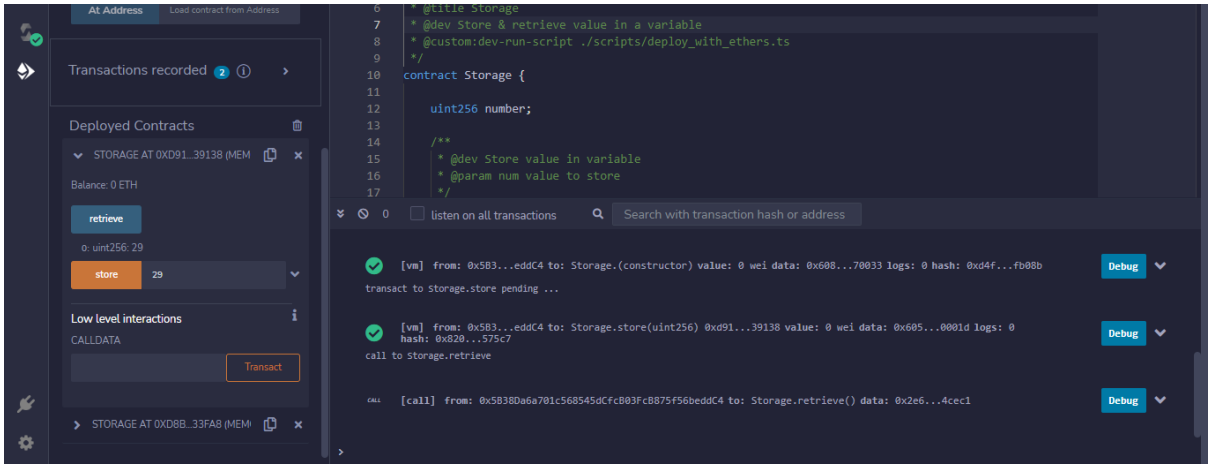
3. You can checkout workspaces also located at left side








ENVIRONMENT
Remix VM (London)
VM
ACCOUNT
0x5B3...eddC4 (99.999999
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT
Storage - contracts/1_Storage.sol
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
pragma solidity >=0.7.0 <0.9.0;
/**
* @title Storage
* @dev Store & retrieve value in a variab
* @custom:dev-run-script ./scripts/deploy
*/
contract Storage {
uint256 number;
/**
* @dev Store value in variable
* @param num value to store
*/
function store(uint256 num) public {
number = num;
}
/**
* @dev Return value
* @return value of 'number'
*/
function retrieve() public view returns
0
listen on all transactions
Search with


At Address
Load contract from Address
Transactions recorded 2
Deployed Contracts
STORAGE AT 0XD91...39138 (MEM
Balance: 0 ETH
retrieve
0: uint256: 29
store
29
Low level interactions
CALLDATA
Transact
STORAGE AT 0XD8B...33FA8 (MEM
* @title Storage
* @dev Store & retrieve value in a variable
* @custom:dev-run-script ./scripts/deploy_with_ethers.ts
*/
contract Storage {
uint256 number;
/**
* @dev Store value in variable
* @param num value to store
*/
0
listen on all transactions
Search with transaction hash or address
[vm] from: 0x5B3...eddC4 to: Storage.(constructor) value: 0 wei data: 0x608...70033 logs: 0 hash: 0xd4f...fb08b
transact to Storage.store pending ...
[vm] from: 0x5B3...eddC4 to: Storage.store(uint256) 0xd91...39138 value: 0 wei data: 0x605...0001d logs: 0
hash: 0x820...575c7
call to Storage.retrieve
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Storage.retrieve() data: 0x2e6...4cec1
Debug
Debug
Debug

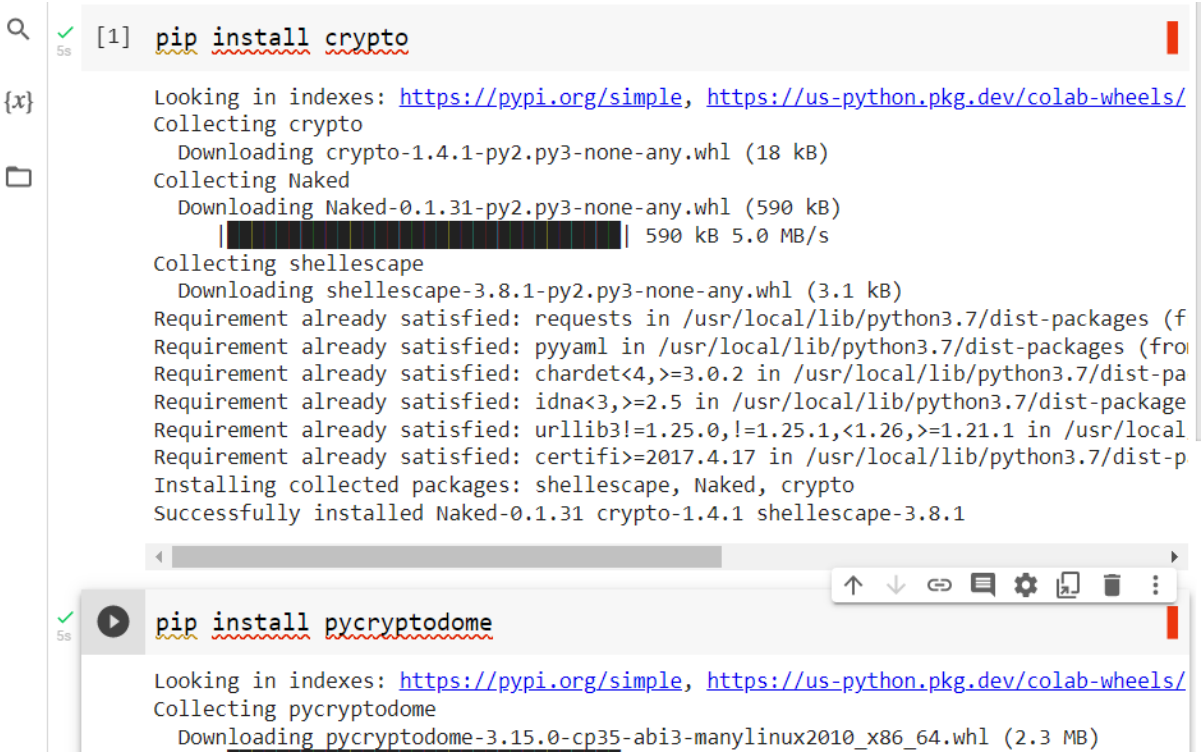
Practical No. 2

Aim: A Simple Client Class that generates the private And public keys by using the built in python RSA algorithm And test.

Theory:

The Rivest-Shamir-Adleman (RSA) encryption algorithm is an asymmetric encryption algorithm that is widely used in many products and services. Asymmetric encryption uses a key pair that is mathematically linked to encrypt and decrypt data. A private and public key are created, with the public key being accessible to anyone and the private key being a secret known only by the key pair creator. With RSA, either the private or public key can encrypt the data, while the other key decrypts it. This is one of the reasons RSA is the most used asymmetric encryption algorithm.

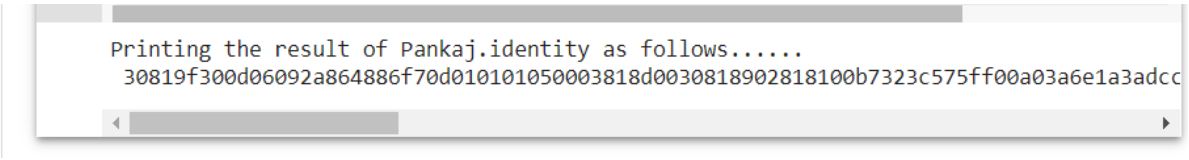
Prerequisite (For Google Colab):



#CODE

```
import Crypto
import binascii
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
class Client:
 def __init__(self):
   random = Crypto.Random.new().read
   self._private_key = RSA.generate(1024, random)
   self._public_key = self._private_key.publickey()
   self._signer = PKCS1_v1_5.new(self._private_key)
 @property
 def identity(self):
   return binascii.hexlify(self._public_key.exportKey(format='
DER')).decode('ascii')
Pankaj = Client()
print ("Printing the result of Pankaj.identity as follows.....
.\n",Pankaj.identity)
```

#Output:-



Practical No: 3

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

Theory: SHA is the acronym for Secure Hash Algorithm, used for hashing data and certificate files. Every piece of data produces a unique hash that is thoroughly non-duplicable by any other piece of data. The resulting digital signature is unique too as it depends on the hash that's generated out of the data

Prerequisite (For Google Colab):

```
[106] pip install crypto

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Requirement already satisfied: crypto in /usr/local/lib/python3.7/dist-packages (1.4
Requirement already satisfied: Naked in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: shellescape in /usr/local/lib/python3.7/dist-packages
```

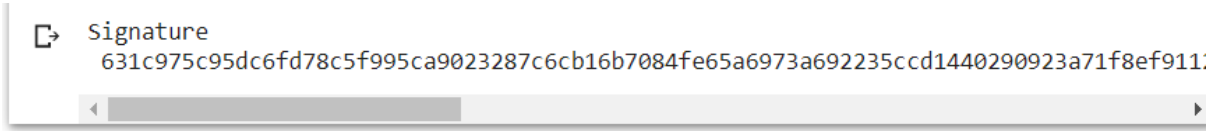
```
[107] pip install pycryptodome

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: pycryptodome in /usr/local/lib/python3.7/dist-pac
```

#Code:

```
import Crypto
import binascii
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
import datetime
import collections
from Crypto.Hash import SHA
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')
class Client:
  def __init__(self):
    random = Crypto.Random.new().read
    self._private_key = RSA.generate(1024, random)
    self._public_key = self._private_key.publickey()
    self._signer = PKCS1_v1_5.new(self._private_key)
  @property
  def identity(self):
    return binascii.hexlify(self._public_key.exportKey(format=
'DER')).decode('ascii')
Pankaj = Client()
Pathak = Client()
t = Transaction(
 Pathak,
 Pankaj.identity,
 5.0)
signature = t.sign_transaction()
print ("Signature\n",signature)
```

Output:-



Practical No: 4

Aim: Create multiple transactions and display them.

Theory:

Transactions are data structures that encode the transfer of value between participants in the bitcoin system. Each transaction is a public entry in bitcoin's blockchain, the global double-entry bookkeeping ledger.

The transactions made by various clients are queued in the system; the miners pick up the transactions from this queue and add it to the block. They will then mine the block and the winning miner would have the privilege of adding the block to the blockchain and thereby earn some money for himself.

Prerequisite (For Google Colab):

```
[106] pip install crypto

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Requirement already satisfied: crypto in /usr/local/lib/python3.7/dist-packages (1.4
Requirement already satisfied: Naked in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: shellescape in /usr/local/lib/python3.7/dist-packages
```

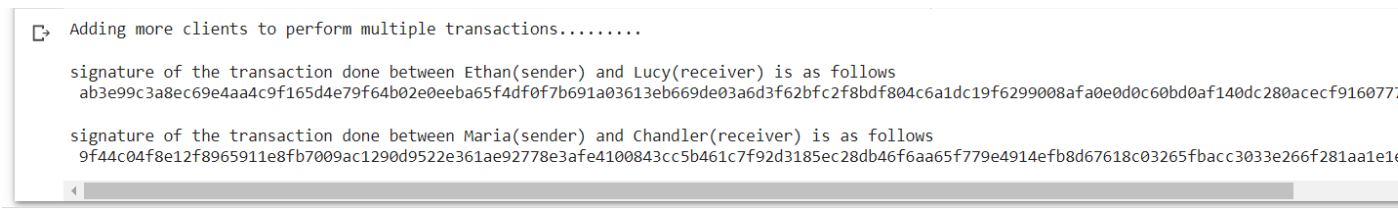
```
[107] pip install pycryptodome

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: pycryptodome in /usr/local/lib/python3.7/dist-pac
```

#CODE

```
import Crypto
import binascii
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
import datetime
import collections
from Crypto.Hash import SHA
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')
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer = PKCS1_v1_5.new(self._private_key)
    @property
    def identity(self):
        return binascii.hexlify(self._public_key.exportKey(forma
t='DER')).decode('ascii')
print("Adding more clients to perform multiple transactions...
......")
Ethan = Client()
Maria = Client()
Lucy = Client()
Chandler = Client()
t1 = Transaction( Ethan, Lucy.identity, 50.0)
signature = t1.sign_transaction()
print("\nsignature of the transaction done between Ethan(sende
r) and Lucy(receiver) is as follows\n",signature)
t2 = Transaction( Maria, Chandler.identity, 25.0)
signature = t2.sign_transaction()
print("\nsignature of the transaction done between Maria(sende
r) and Chandler(receiver) is as follows\n",signature)
```

Output:-



Practical No: 5

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

Theory:

- **Blockchain:** A blockchain is a type of distributed ledger technology (DLT) that consists of growing list of records, called blocks, that are securely linked together using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data nodes are represented by leafs). The timestamp proves that the transaction data existed when the block was created. Since each block contains information about the block previous to it, they effectively form a chain (compare linked list data structure), with each additional block linking to the ones before it. Consequently, blockchain transactions are irreversible in that, once they are recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks. Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collectively adhere to a consensus algorithm protocol to add and validate new transaction blocks. Although blockchain records are not unalterable, since blockchain forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.
- **Genesis:** block The genesis block is the first block in any blockchain-based protocol. It is the basis on which additional blocks are added to form a chain of blocks, hence the term blockchain. This block is sometimes referred to Block 0. Every block in a blockchain stores a reference to the previous block. In the case of Genesis Block, there is no previous block for reference. In technical terms, it means that the Genesis Block has it's "previous hash" value set to 0. This means that no data was processed before the Genesis Block. All other blocks will have sequential numbers starting by 1, and will have a "previous hash" set to the hash of the previous block.

Code:

```
import hashlib
import datetime

class Block:
  def __init__(self, previous_block_hash, data, timestamp):
    self.previous_block_hash = previous_block_hash
    self.data = data
    self.timestamp = timestamp
    self.hash = self.get_hash()
  @staticmethod
  def create_genesis_block():
    return(Block("0", "0", datetime.datetime.now()))
  def get_hash(self):
    header = (str(self.previous_block_hash) +str(self.data) +s
tr(self.timestamp))
    inner_hash = hashlib.sha256(header.encode()).hexdigest().e
ncode()
```

```
    comp_hash = hashlib.sha256(inner_hash).hexdigest()
    return comp_hash
number_of_blocks = 14
Blockchain = [Block.create_genesis_block()]
print("Genesis Block is Created")
print("Hash: %s" % Blockchain[0].hash)
for i in range(1, number_of_blocks):
  Blockchain.append(Block(Blockchain[i-
1].hash,"Block number %d" %i, datetime.datetime.now()))
  print("%d block created" %i)
  print("Hash: %s" % Blockchain[-1].hash)
```

Output:-

```
Genesis Block is Created
Hash: f044a494f5b02a9507d70d8e9b15eccee589f0a49d3436e936d831d11234e0a9
1 block created
Hash: 3129f4baebe87613fe5bf718db6675021cffbc9498bc664b1bdfbfd3f7e8a247
2 block created
Hash: 0aa2a506056ef29ef950f07ab877ff4e029b79c2ccb7906b7ad45eb19e50e514
3 block created
Hash: 50fde4d261d2733197a0a6bc3806bc89675a687c7b6f1f3d3be4a7a5ceb18cec
4 block created
Hash: 572d02c78a2b02446b0d8b212f68f46bd2ddba18cd2237638f63c5ae022fad5e
5 block created
Hash: 41bc69643e66f4bbccec90300d35135689550c1ef20f7b7421bf6eb938307ebd
6 block created
Hash: 131b0448f0c2ca6b8eac8f5e5bd7ae1afe00d3242ff0fdd0420a43fa3d28073c
7 block created
Hash: 98ebb788267c33d77259a0ab4056c176f6dc0070cce4219720453c2450df918a
8 block created
Hash: cf5d089370ad649d426d4e64c205d85c27ce39a7e54879fc6cd3819d53f2d5d9
9 block created
Hash: c7f64c1b6855afd23cd942284cbfd2d15bcf5187602f0b96be9edf3fd1b2f783
10 block created
Hash: aa4376e5408af77b5dca571a0c6770bef4cd3b0b9ba4f402bcdf0effb69f7480
11 block created
Hash: e3f27e122bdb13dba3c87545e54462f9125a525bf9f0c34468cd94b20806fcf9
12 block created
Hash: 2a52d062cf5bfbb7f86887bbe1b4f79db72fd9bb03f4803bf97423435166c7e9
13 block created
Hash: e3abe5be7bc5e3b7fc1d7dd7ed871c481efcececc9a32dd1b9c6f40cf4781e65
```

Practical No: 6

Aim: Create a mining function and test it.

Theory:

Mining: Mining is the process that Bitcoin and several other cryptocurrencies use to generate new coins and verify new transactions. It involves vast, decentralized networks of computers around the world that verify and secure blockchains – the virtual ledgers that document cryptocurrency transactions. In return for contributing their processing power, computers on the network are rewarded with new coins. It's a virtuous circle: the miners maintain and secure the blockchain, the blockchain awards the coins, the coins provide an incentive for the miners to maintain the blockchain.

Code:

```
from hashlib import sha256
MAX_NONCE = 100000000000
def SHA256(text):
  return sha256(text.encode("ascii")).hexdigest()
def mine(block_number, transactions, previous_hash, prefix_zeros):
  prefix_str = '0' *prefix_zeros
  for nonce in range(MAX_NONCE):
    text = str(block_number) + transactions + previous_hash + str(nonce)
    new_hash = SHA256(text)
    if new_hash.startswith (prefix_str):
      print(f"Yay! Successfully mined bitcoins with nonce value:{ nonce}")
    return new_hash
  raise BaseException(f"Couldn't find correct has after trying (MAX_NONCE)
limes")
  if __name__ == "__main__":
    transactions='''
    Dhaval->Bhavin->20,
    Mando->Cara->45
    '''
    difficulty=4
import time
start = time.time()
print("start mining")
new_hash = mine(5, transactions, '0000000xa036944e29568d0cff17edbe038f81208
fecf9a66be9a2b8321c6ec7', difficulty)

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

Output:-

```
start mining
Yay! Successfully mined bitcoins with nonce value:56998
end mining. Mining took: 0.10538816452026367 seconds
0000f66fd90fe47408579671818546c0a603f8c921b5c9c525906ec1bac11800
```

Practical No: 7

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

Theory: A blockchain is a decentralized ledger of all transactions across a peer-to-peer network. Using this technology, participants can confirm transactions without a need for a central clearing authority.

Code:-

```
import datetime
import hashlib
class Block:
  blockNo = 0
  data = None
  next = None
  hash = None
  nonce = 0
  previous_hash = 0x0
  timestamp = datetime.datetime.now()
  def __init__(self, data):
    self.data = data
  def hash(self):
    h = hashlib.sha256()
    h.update(
        str(self.nonce).encode('utf-8') +
        str(self.data).encode('utf-8') +
        str(self.previous_hash).encode('utf-8') +
        str(self.timestamp).encode('utf-8') +
        str(self.blockNo).encode('utf-8')
        )
    return h.hexdigest()
  def __str__(self):
    return "Block Hash: " + str(self.hash()) + "\nBlockNo: " +
 str(
self.blockNo) + "\nBlock Data: " + str(self.data) + "\nHashes:
 " + str(
self.nonce) + "\n--------------"
class Blockchain:
  diff = 20
  maxNonce = 2**32
  target = 2 ** (256-diff)
  block = Block("Genesis")
  dummy = head = block
  def add(self, block):
    block.previous_hash = self.block.hash()
```

```
    block.blockNo = self.block.blockNo + 1
    self.block.next = block
    self.block = self.block.next
  def mine(self, block):
   for n in range(self.maxNonce):
     if int(block.hash(), 16) <= self.target:
       self.add(block)
       print(block)
       break
     else:
       block.nonce += 1
blockchain = Blockchain()
for n in range(10):
  blockchain.mine(Block("Block " + str(n+1)))
while blockchain.head != None:
  print(blockchain.head)
  blockchain.head = blockchain.head.next
```

Output:-

```
Block Hash:
67a42937b01a3328aca1ee685e26bc8ef192e3de8584142892a9a78d43dfcbac
BlockNo: 1
Block Data: Block 1
Hashes: 3630387
--------------
Block Hash:
1ad1df93868a7740560acf71938343412afd8c88283047971fecbcd72dc6a24f
BlockNo: 2
Block Data: Block 2
Hashes: 358818
--------------
Block Hash:
d0df8bd38f9479970e3e51a7c70b353b2163b38866a2146c74f604c23a439dce
BlockNo: 3
Block Data: Block 3
Hashes: 3229484
--------------
Block Hash:
a82629bf4f0648c71145491f6054d814fe7d03bd56e0d0c11cd082e139e510e0
BlockNo: 4
Block Data: Block 4
Hashes: 4435
--------------
Block Hash:
2fff7b1ea679d975e3e2668239a0054fd6e44e291ebd652a632c91924dfe17c9
BlockNo: 5
Block Data: Block 5
Hashes: 294780
--------------
```

```
Block Hash:
c8bbf6be119e90fd4baeba2744f6b412f02e46d95c0c4ab5933b690875c6eca9
BlockNo: 6
Block Data: Block 6
Hashes: 460826
--------------
Block Hash:
4edb1d96832edd6fb6bc9ffa6b9fda2c395ec6983e2c99da5deaf7b7c468a7e4
BlockNo: 7
Block Data: Block 7
Hashes: 1929340
--------------
Block Hash:
45ed9b2f49c20cabbf5aa734f46a3dbae93fc4618418e108771683b2bbf00512
BlockNo: 8
Block Data: Block 8
Hashes: 1665985
--------------
Block Hash:
a7f0d3e4c34871db193a77c47d39fcb44c9324e8dcbd69e25c51efdf5c77a6f8
BlockNo: 9
Block Data: Block 9
Hashes: 361489
--------------
Block Hash:
18235b03dc74c5e6b010e4355a47dadceeb14803ff6721a2567b013d672d0330
BlockNo: 10
Block Data: Block 10
Hashes: 177757
--------------
Block Hash:
a3fac650f838b387c77afbf1c13bf4d4df68d6e5c541c9d412d7d550e4e284c9
BlockNo: 0
Block Data: Genesis
Hashes: 0
--------------
Block Hash:
67a42937b01a3328aca1ee685e26bc8ef192e3de8584142892a9a78d43dfcbac
BlockNo: 1
Block Data: Block 1
Hashes: 3630387
--------------
Block Hash:
1ad1df93868a7740560acf71938343412afd8c88283047971fecbcd72dc6a24f
BlockNo: 2
Block Data: Block 2
Hashes: 358818
--------------
Block Hash:
d0df8bd38f9479970e3e51a7c70b353b2163b38866a2146c74f604c23a439dce
BlockNo: 3
Block Data: Block 3
Hashes: 3229484
--------------
Block Hash:
a82629bf4f0648c71145491f6054d814fe7d03bd56e0d0c11cd082e139e510e0
BlockNo: 4
Block Data: Block 4
```

```
Hashes: 4435
--------------
Block Hash:
2fff7b1ea679d975e3e2668239a0054fd6e44e291ebd652a632c91924dfe17c9
BlockNo: 5
Block Data: Block 5
Hashes: 294780
--------------
Block Hash:
c8bbf6be119e90fd4baeba2744f6b412f02e46d95c0c4ab5933b690875c6eca9
BlockNo: 6
Block Data: Block 6
Hashes: 460826
--------------
Block Hash:
4edb1d96832edd6fb6bc9ffa6b9fda2c395ec6983e2c99da5deaf7b7c468a7e4
BlockNo: 7
Block Data: Block 7
Hashes: 1929340
--------------
Block Hash:
45ed9b2f49c20cabbf5aa734f46a3dbae93fc4618418e108771683b2bbf00512
BlockNo: 8
Block Data: Block 8
Hashes: 1665985
--------------
Block Hash:
a7f0d3e4c34871db193a77c47d39fcb44c9324e8dcbd69e25c51efdf5c77a6f8
BlockNo: 9
Block Data: Block 9
Hashes: 361489
--------------
Block Hash:
18235b03dc74c5e6b010e4355a47dadceeb14803ff6721a2567b013d672d0330
BlockNo: 10
Block Data: Block 10
Hashes: 177757
--------------
```

Practical No: 8

Aim: Implement and Demonstrate the Use of Solidity Programming:

A) Your First Solidity Smart Contract (Counter Program)

B) To Create and Explore Types of Variables with Varying Data Types in Solidity Programming (Variables).

Theory:

Solidity :

Solidity is an object-oriented programming language created specifically by the Ethereum Network team for constructing and designing smart contracts on Blockchain platforms.

- It's used to create smart contracts that implement business logic and generate a chain of transaction records in the blockchain system.
- It acts as a tool for creating machine-level code and compiling it on the Ethereum Virtual Machine (EVM).
- It has a lot of similarities with C and C++ and is pretty simple to learn and understand. For example, a “main” in C is equivalent to a “contract” in Solidity.
- Like other programming languages, Solidity programming also has variables, functions, classes, arithmetic operations, string manipulation, and many other concepts.

Smart Contracts :

- Smart contracts refer to high-level program codes compiled into EVM before being posted to the Ethereum blockchain for execution.
- It enables you to conduct trustworthy transactions without the involvement of a third party; these transactions are traceable and irreversible.
- Programming languages commonly used to create and write smart contracts are Serpent, Solidity, Mutan, and LLL.

Value Types :

Value type variables store their own data. These are the basic data types provided by solidity. These types of variables are always passed by value. The variables are copied wherever they are used in function arguments or assignment. Value type data types in solidity are listed below

- **Boolean:** This data type accepts only two values True or False.
- **Integer**: This data type is used to store integer values, int and uint are used to declare signed and unsigned integers respectively.
- **Fixed Point Numbers:** These data types are not fully supported in solidity yet, as per the Solidity documentation. They can be declared as fixed and unfixed for signed and unsigned fixed-point numbers of varying sizes respectively.
- **Address:** Address hold a 20-byte value which represents the size of an Ethereum address. An address can be used to get balance or to transfer a balance by balance and transfer method respectively.
- **Bytes and Strings:** Bytes are used to store a fixed-sized character set while the string is used to store the character set equal to or more than a byte. The length of bytes is from 1 to 32,

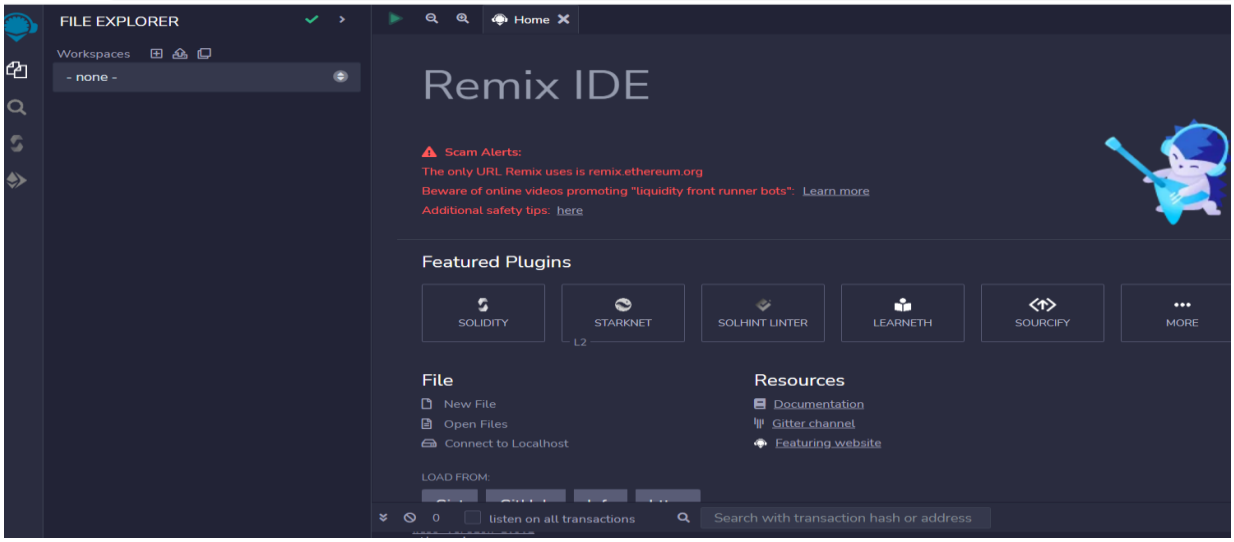
while the string has a dynamic length. Byte has an advantage that it uses less gas, so better to use when we know the length of data.

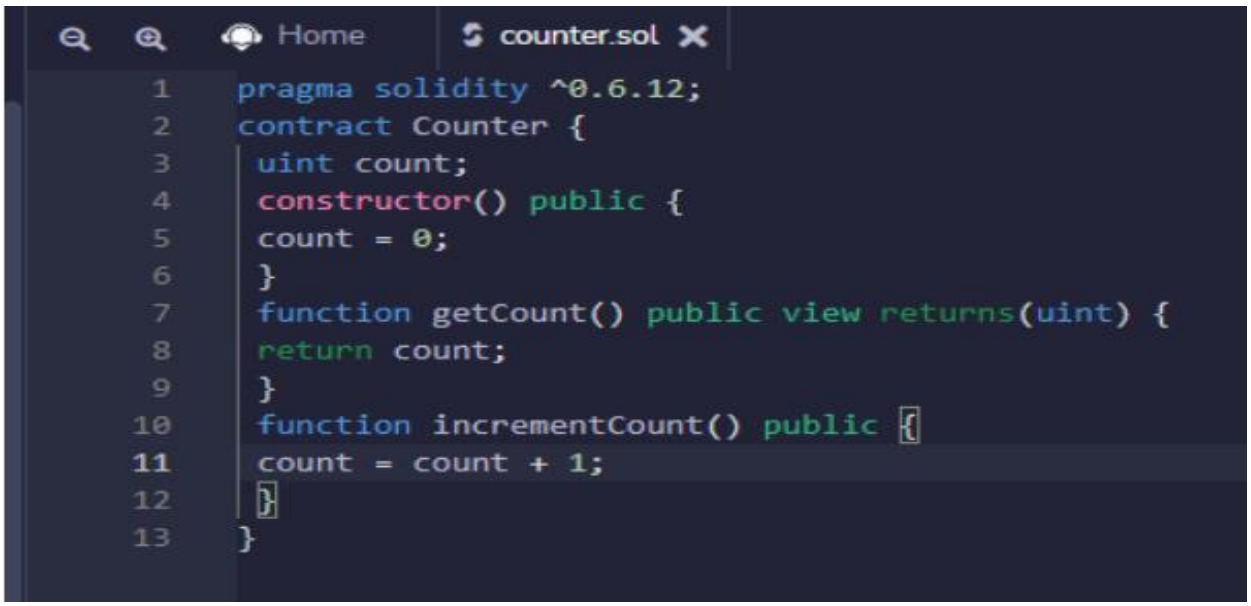
- **Enums:** It is used to create user-defined data types, used to assign a name to an integral constant which makes the contract more readable, maintainable, and less prone to errors. Options of enums can be represented by unsigned integer values starting from 0.

Steps:

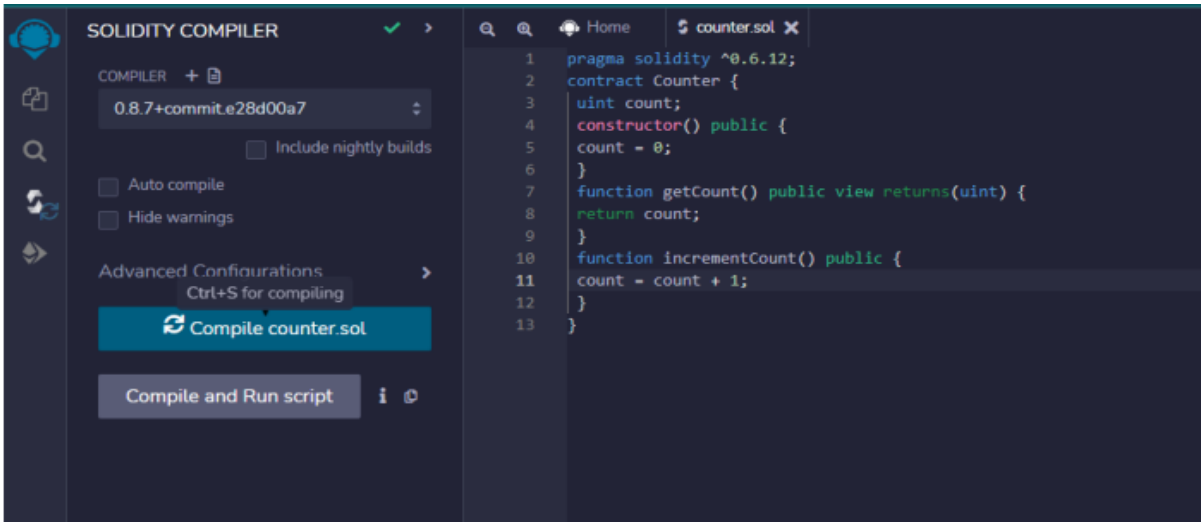
A. Your First Solidity Smart Contract (Counter Program)

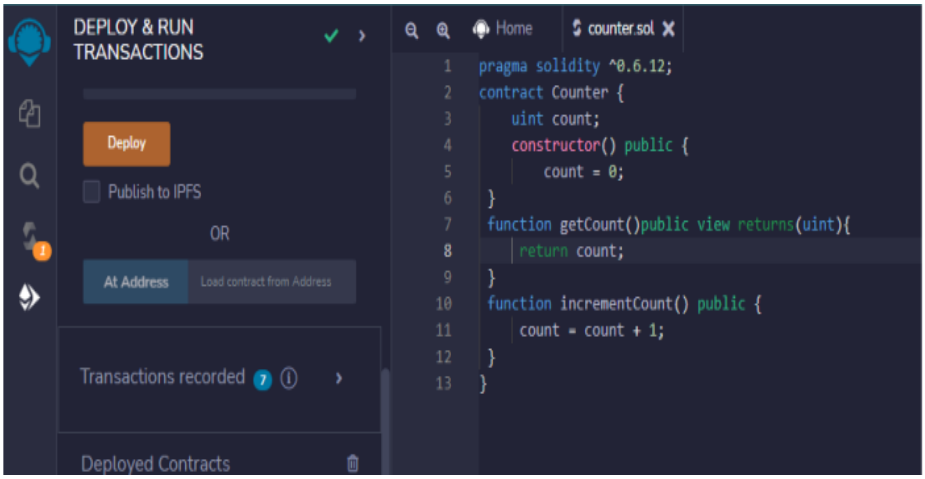
1. Go to https://remix.ethereum.org/ after that click on “create new file” in default workspace session and create the file name counter.sol and add below given code



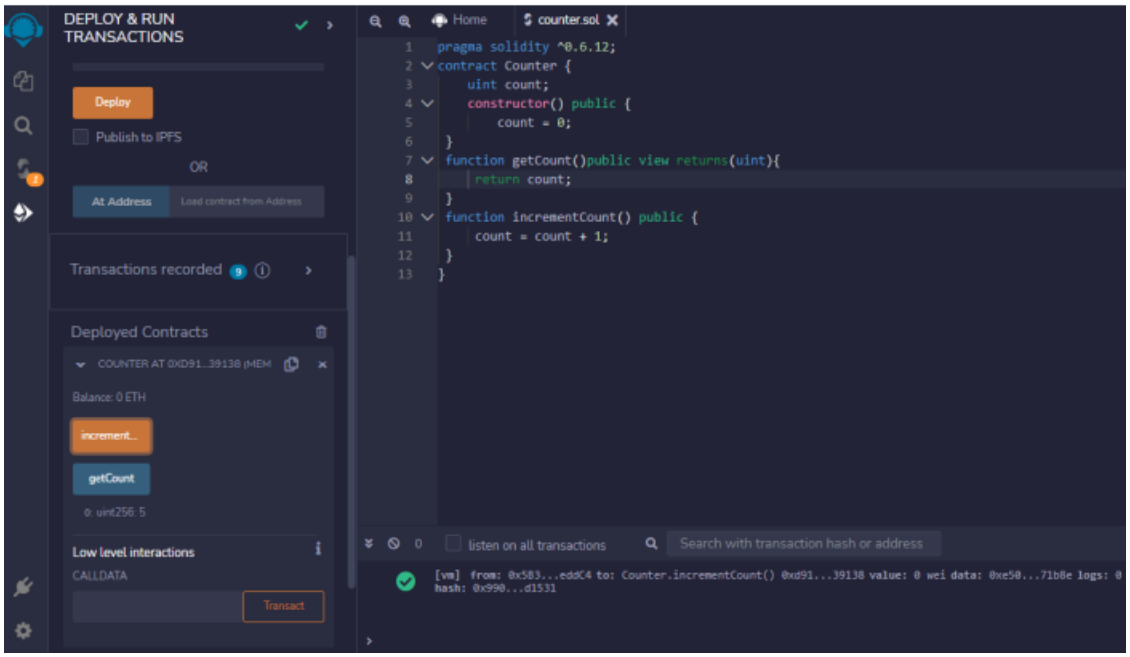


2. Now Compile and go to Deploy and run transaction and click on deploy program

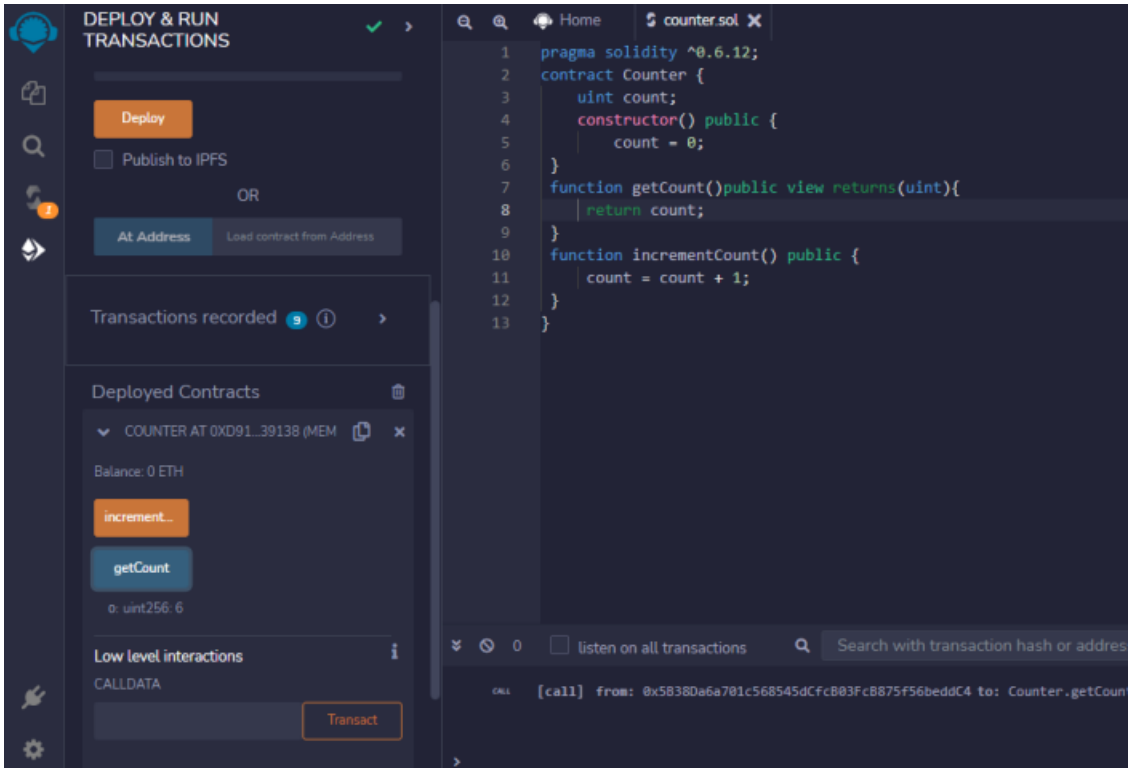

SOLIDITY COMPILER
COMPILER
0.8.7+commit.e28d00a7
Include nightly builds
Auto compile
Hide warnings
Advanced Configurations
Ctrl+S for compiling
Compile counter.sol
Compile and Run script
Home
counter.sol
pragma solidity ^0.6.12;
contract Counter {
uint count;
constructor() public {
count = 0;
}
function getCount() public view returns(uint) {
return count;
}
function incrementCount() public {
count = count + 1;
}
}


DEPLOY & RUN TRANSACTIONS
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 7
Deployed Contracts
Home
counter.sol
pragma solidity ^0.6.12;
contract Counter {
uint count;
constructor() public {
count = 0;
}
function getCount()public view returns(uint){
return count;
}
function incrementCount() public {
count = count + 1;
}
}

Output:-


DEPLOY & RUN TRANSACTIONS
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 9
Deployed Contracts
COUNTER AT 0XD91...39138 (MEM
Balance: 0 ETH
increment...
getCount
0: uint256: 5
Low level interactions
CALLDATA
Transact
Home
counter.sol
pragma solidity ^0.6.12;
contract Counter {
uint count;
constructor() public {
count = 0;
}
function getCount()public view returns(uint){
return count;
}
function incrementCount() public {
count = count + 1;
}
}
listen on all transactions
Search with transaction hash or address
[vm] from: 0x583...eddC4 to: Counter.incrementCount() 0xd91...39138 value: 0 wei data: 0xe50...71b8e logs: 0
hash: 0x990...d1531

Deployed Contracts:



B. To Create and Explore Types of Variables with Varying Data Types in Solidity Programming (Variables)

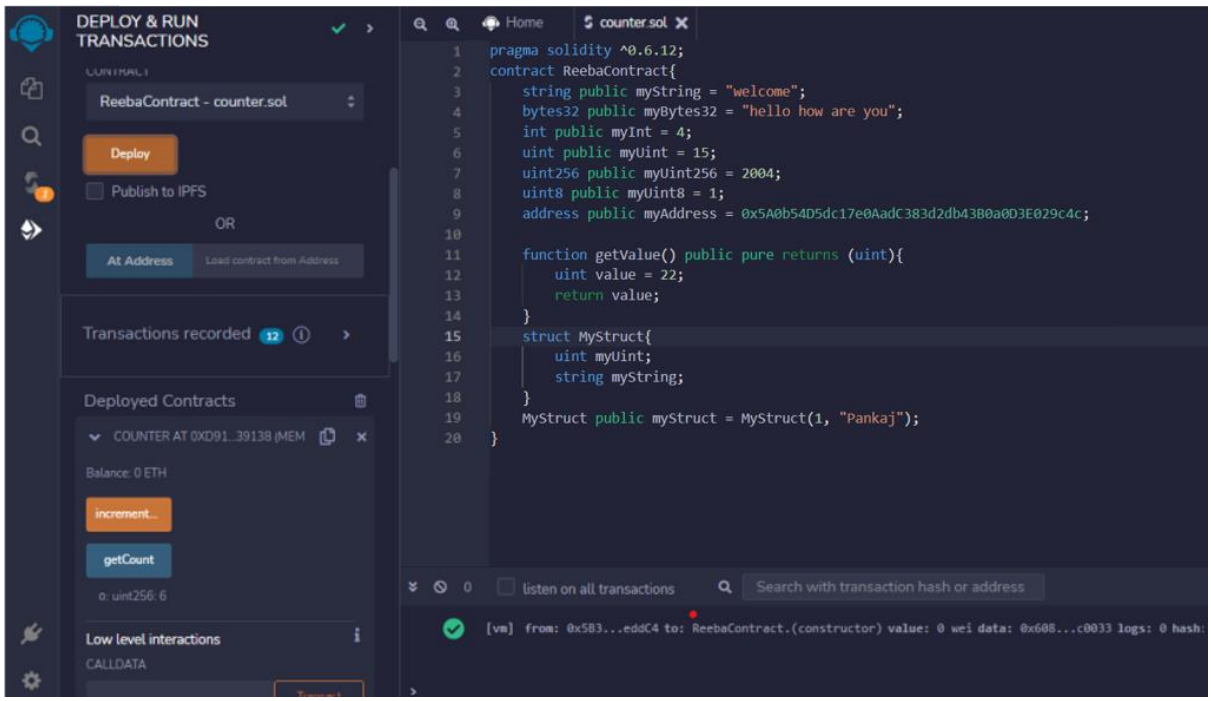
File name as variable.sol

CODE:-

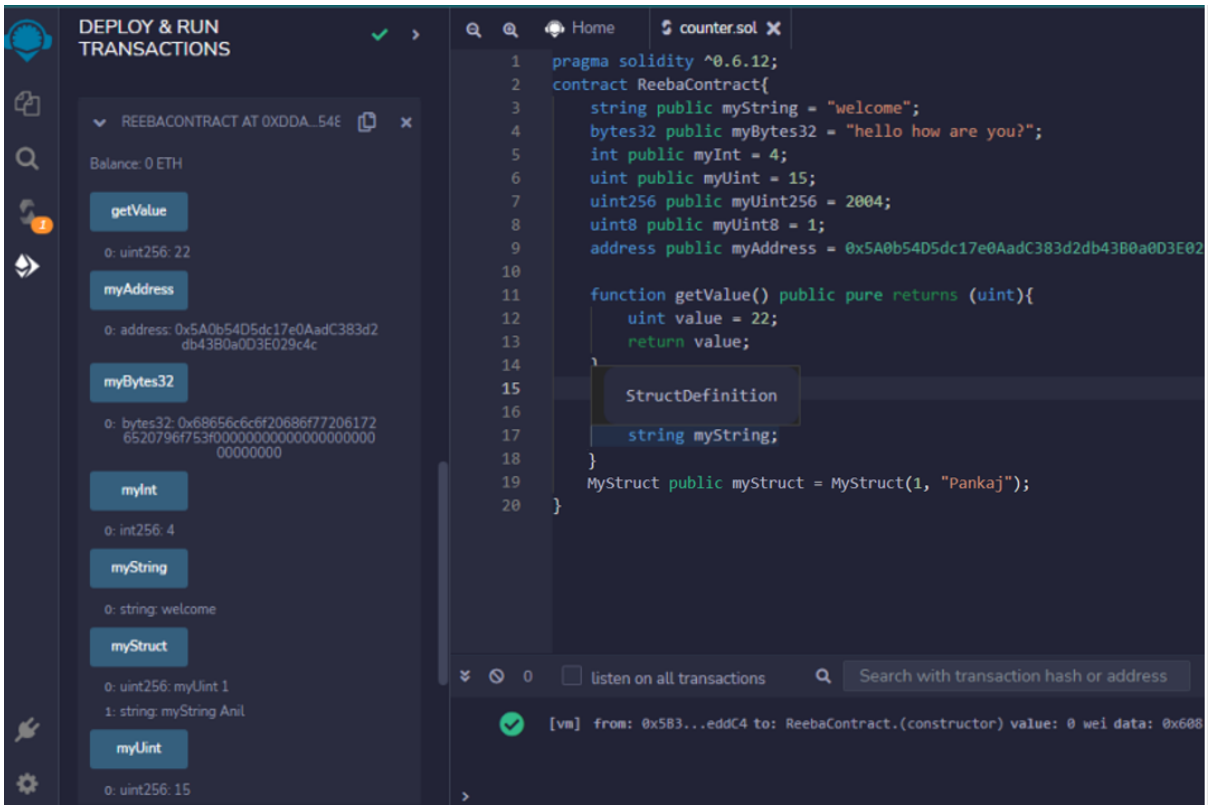
```
pragma solidity ^0.6.12;
contract ReebaContract{
    string public myString = "welcome";
    bytes32 public myBytes32 = "hello how are you";
    int public myInt = 4;
    uint public myUint = 15;
    uint256 public myUint256 = 2004;
    uint8 public myUint8 = 1;
    address public myAddress = 0x5A0b54D5dc17e0AadC383d2db43B0a0D3E029c4c;

    function getValue() public pure returns (uint){
        uint value = 22;
        return value;
    }
    struct MyStruct{
        uint myUint;
        string myString;
    }
    MyStruct public myStruct = MyStruct(1, "Pankaj");
}
```

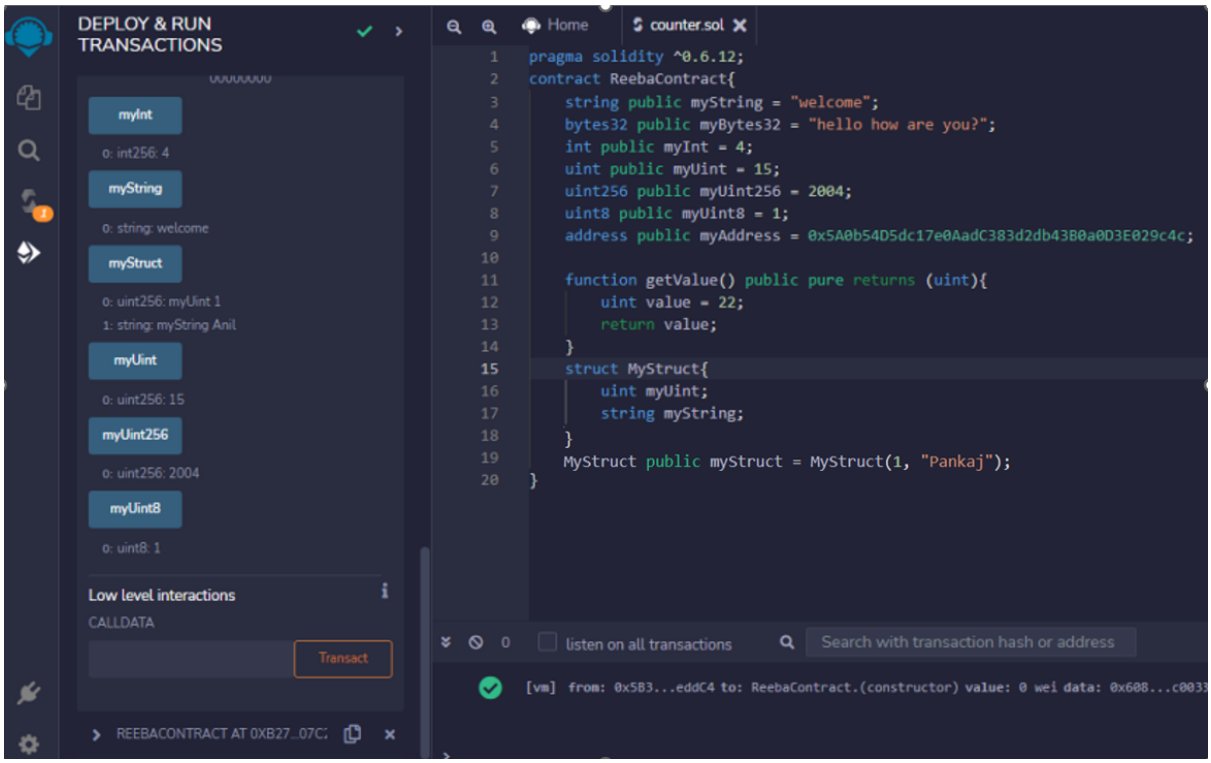
Output:

Deploy output:



Deployed Contracts:





Practical No: 9

Aim: Develop a decentralized voting application using solidity

Theory: We are creating a Decentralized Voting Application (DApps) which is built on Solidity Language. This Project showcases a lot of Solidity's features. It implements a voting contract.

Code:

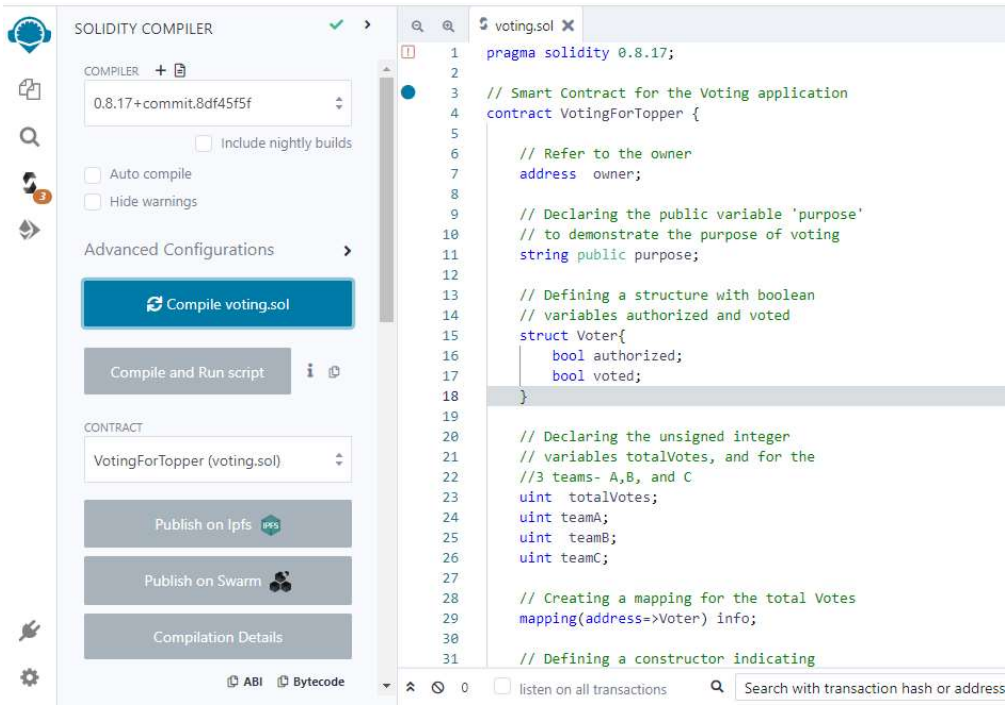
```
pragma solidity 0.8.17;

// Smart Contract for the Voting application
contract VotingForTopper {

    // Refer to the owner
    address  owner;

    // Declaring the public variable 'purpose'
    // to demonstrate the purpose of voting
    string public purpose;

    // Defining a structure with boolean
    // variables authorized and voted
    struct Voter{
        bool authorized;
        bool voted;
    }

    // Declaring the unsigned integer
    // variables totalVotes, and for the
    //3 teams- A,B, and C
    uint  totalVotes;
    uint teamA;
    uint  teamB;
    uint teamC;

    // Creating a mapping for the total Votes
    mapping(address=>Voter) info;

    // Defining a constructor indicating
    // the purpose of voting
    constructor(string memory _name) public{
        purpose = _name;
        owner = msg.sender;
    }

    // Defining a modifier to
    // verify the ownership
    modifier ownerOn() {
```

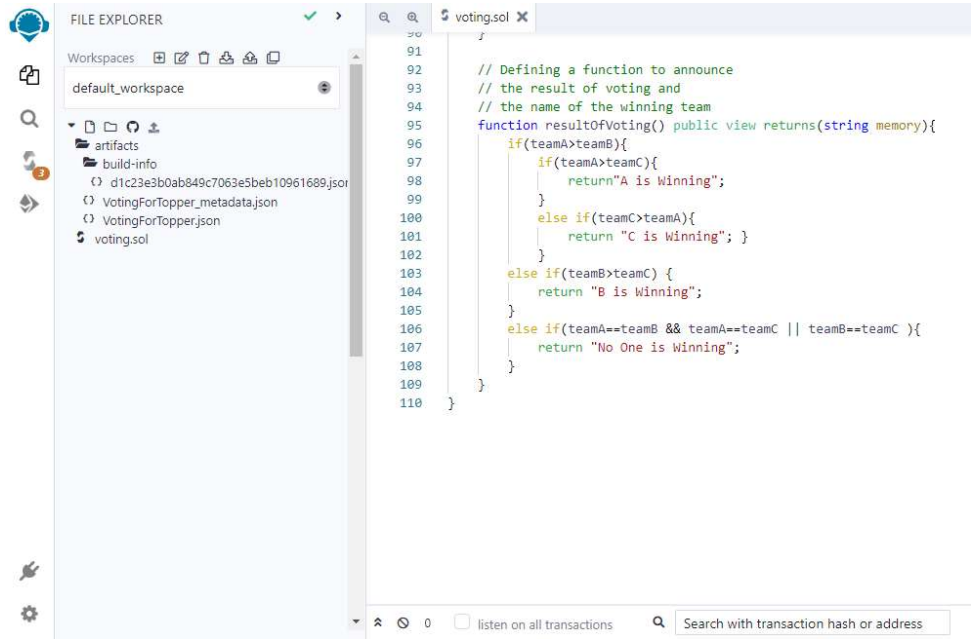
```
        require(msg.sender==owner);
        _;
    }

    // Defining a function to verify
    // the person is voted or not
    function authorize(address _person) ownerOn public {
        info[_person].authorized= true;
    }

    // Defining a function to check and
    // skip the code if the person is already
    // voted else allow to vote and
    // calculate totalvotes for team A
    function temaAF(address _address) public {
        require(!info[_address].voted,"already voted person");
        require(info[_address].authorized,"You Have No Right for Vote");
        info[_address].voted = true;
        teamA++;
        totalVotes++;
    }

    // Defining a function to check
    // and skip the code if the person
    // is already voted else allow to vote
    // and calculate totalvotes for team B
    function temaBF(address _address) public {
        require(!info[_address].voted,"already voted person");
        require(info[_address].authorized,"You Have No Right for Vote");
        teamB++;
        info[_address].voted = true;
        totalVotes++;
    }

    // Defining a function to check
    // and skip the code if the person
    // is already voted else allow to vote
    // and calculate totalvotes for team C
    function temaCF(address _address) public returns(string memory){
        require(!info[_address].voted,"already voted person");
        require(info[_address].authorized,"You Have No Right for Vote");
        info[_address].voted = true;
        teamC++;
        totalVotes++;
        return("Thanks for Voting");
    }

    function totalVotesF() public view returns(uint){
```

```
        return totalVotes;
    }

    // Defining a function to announce
    // the result of voting and
    // the name of the winning team
    function resultOfVoting() public view returns(string memory){
        if(teamA>teamB){
            if(teamA>teamC){
                return"A is Winning";
            }
            else if(teamC>teamA){
                return "C is Winning"; }
            }
        else if(teamB>teamC) {
            return "B is Winning";
        }
        else if(teamA==teamB && teamA==teamC || teamB==teamC ){
            return "No One is Winning";
        }
    }
}
```

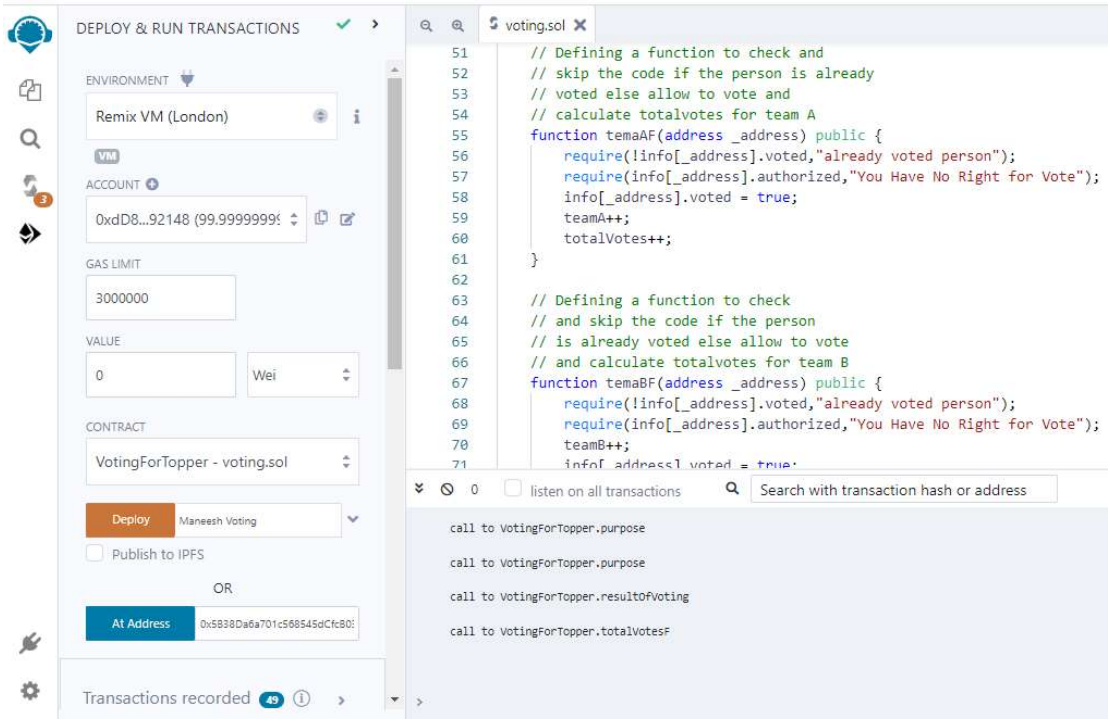
Output:

Compile the code





After Compile go to Deploy & run transactions copy account key from account session and deploy the program



Now go to Deployed Contracts menu at the bottom of the Deploy & run transactions. Paste the key in front of authorize button and click on authorize > teamAF > teamBF > teamCF > purpose > result of voting > totalVotes