

Hindi Vidya Prachar Samiti's
**RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE &
COMMERCE**
(EMPOWERED AUTONOMOUS COLLEGE)



AFFILIATED TO
UNIVERSITY OF MUMBAI

DEPARTMENT OF INFORMATION TECHNOLOGY
2024-2025

M.Sc. (IT) PART I SEM II

**PAPER RJSPI203P – THEORY AND APPLICATIONS OF
BLOCKCHAIN**

Name :- Vishwakarma Shivam Suresh Sushila

Roll No. 6620

Hindi Vidya Prachar Samiti's
RAMNIRANJAN JHUNJHUNWALA COLLEGE
Ghatkopar (W), Mumbai-400 086

Certificate



This is to certify that Mr./Ms. Vishwakarma Shivam Purosh Suhila Roll No 6620 of M.Sc.(I.T) Part-1 class has completed the required number of experiments in the subject of Theory and Applications of Blockchain in the Department of Information Technology during the academic year 2024 - 2025.

Professor In-Charge

Co-ordinator of IT Department
Prof. Bharati Bhole

College Seal & Date

Examiner

INDEX

Sr. No	Title	Date
1	Demonstrate Working with MetaMask a. Install Metamask, Create account, Deposit Ether. b. Transfer ether in between accounts and observe the transaction details.	Nov 25, 2024
2	Implementing cryptography Algorithm in Python: a. DES b. SHA Message Digest	Nov 27, 2024
3	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 .	Dec 2, 2024
4	Write the following programs for Blockchain in Python : a. Create multiple transactions and display them . b. Create a blockchain, a genesis block and execute it .	Dec 4, 2024
5	Write the following programs for Blockchain in Python : a. Create a mining function and test it. b. Add blocks to the miner and dump the blockchain.	Dec 11, 2024
6	Write the solidity Code and demonstrate the following: a. Variable b. Operators c. Decision Making - if, if else, if else if d. Loops - for, while, and do while	Dec 18, 2024
7	Write the solidity Code and demonstrate the following: a. Functions b. View Functions c. Pure Functions d. Cryptographic Function	Jan 1, 2025
8	Write the solidity Code and demonstrate the following: a. Contracts b. Inheritance c. Constructors	Jan 8, 2025

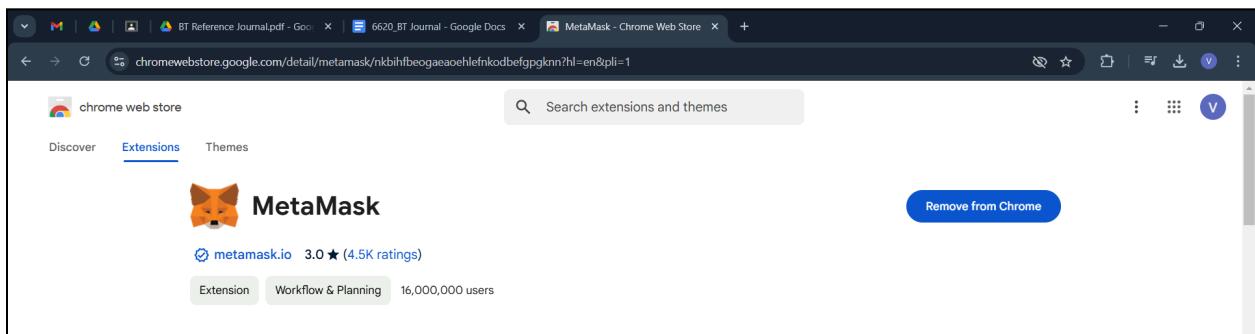
	d. Interfaces	
9	Program a Simple contract that can get, increment and decrement the count store in the contract.	Jan 15, 2025
10	<p>Write a simple smart contract to calculate the ‘number of ethers’ for the transaction of gas limit for the given scenario.</p> <ul style="list-style-type: none"> a. Mint Your Own Coins and Make Simple Transactions With Solidity b. Creating ERC -20 Token 	Jan 22, 2025

Practical 1: Demonstrate Working with MetaMask

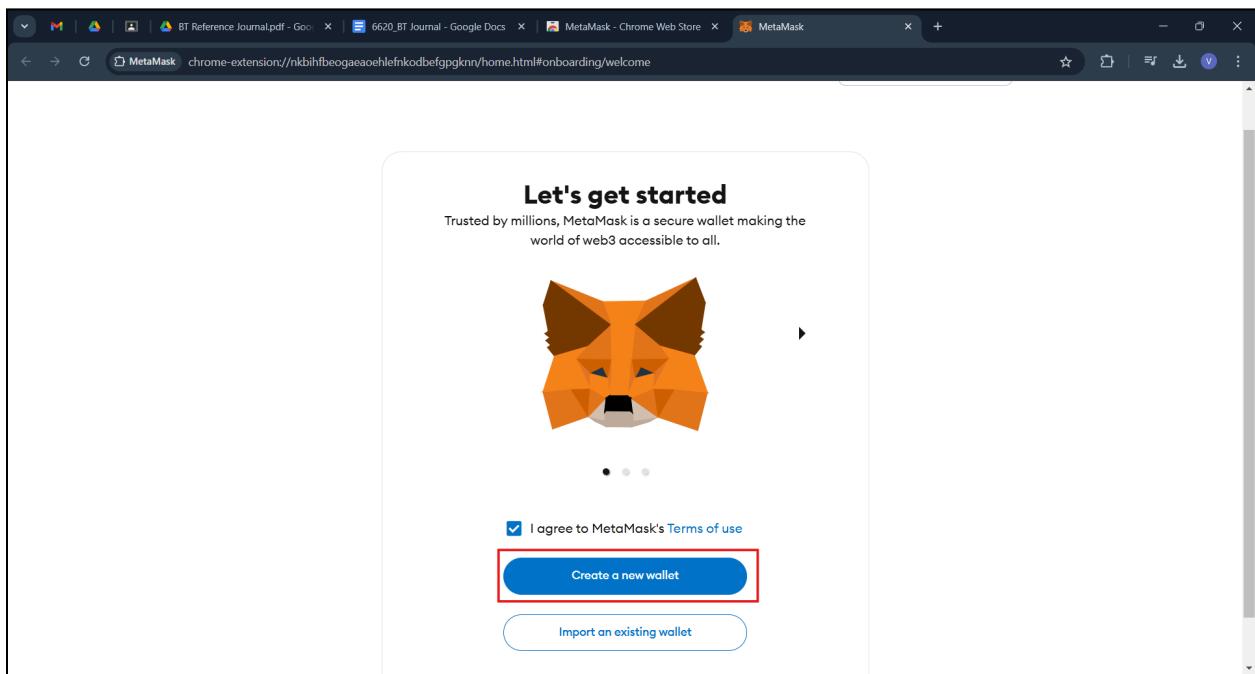
- a. Install Metamask ,Create account ,Deposit Ether
 - b. Transfer ether in between accounts and observe the transaction details
-

A. Install Metamask, Create account, Deposit Ether

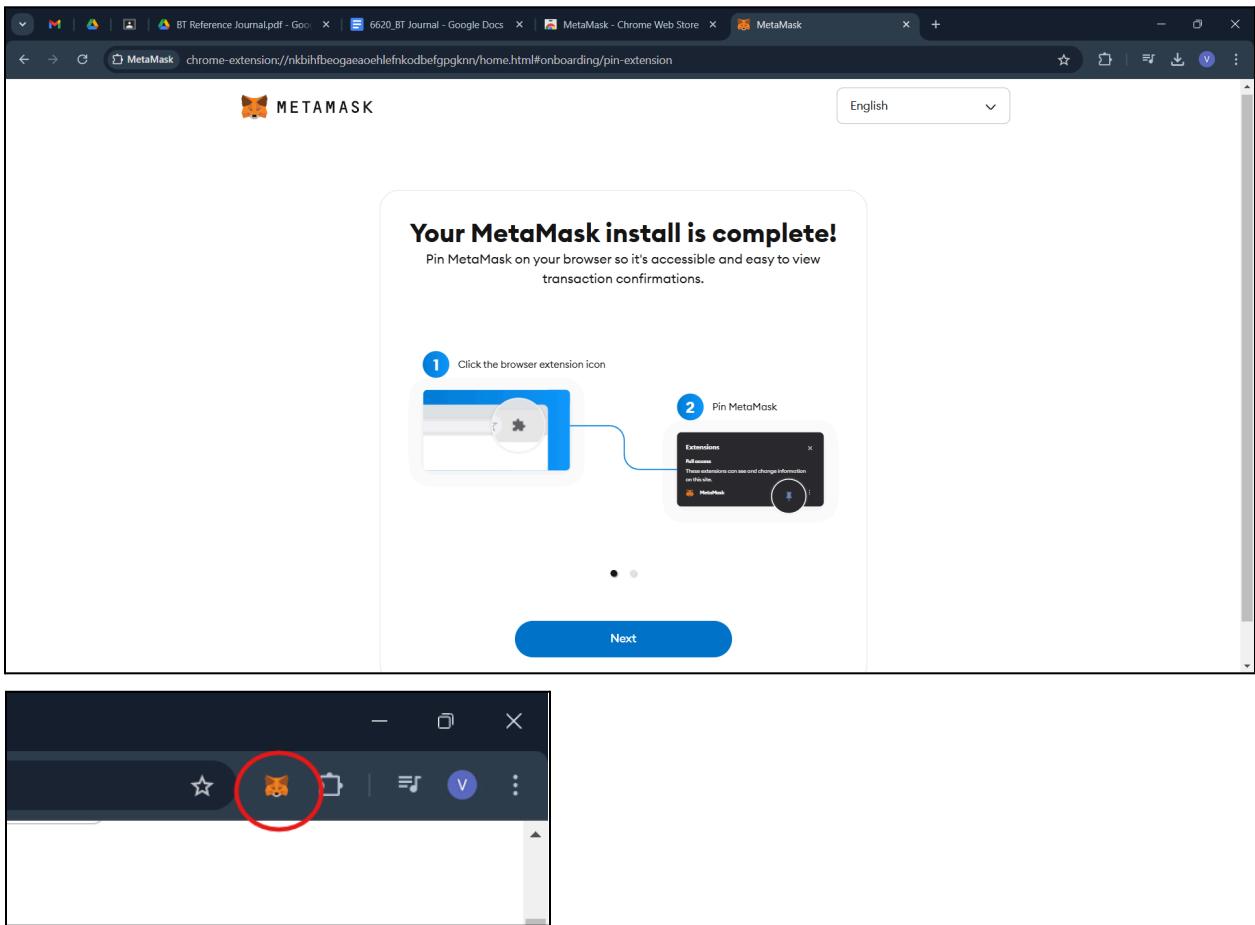
To Install search Metamask extension and add to chrome.



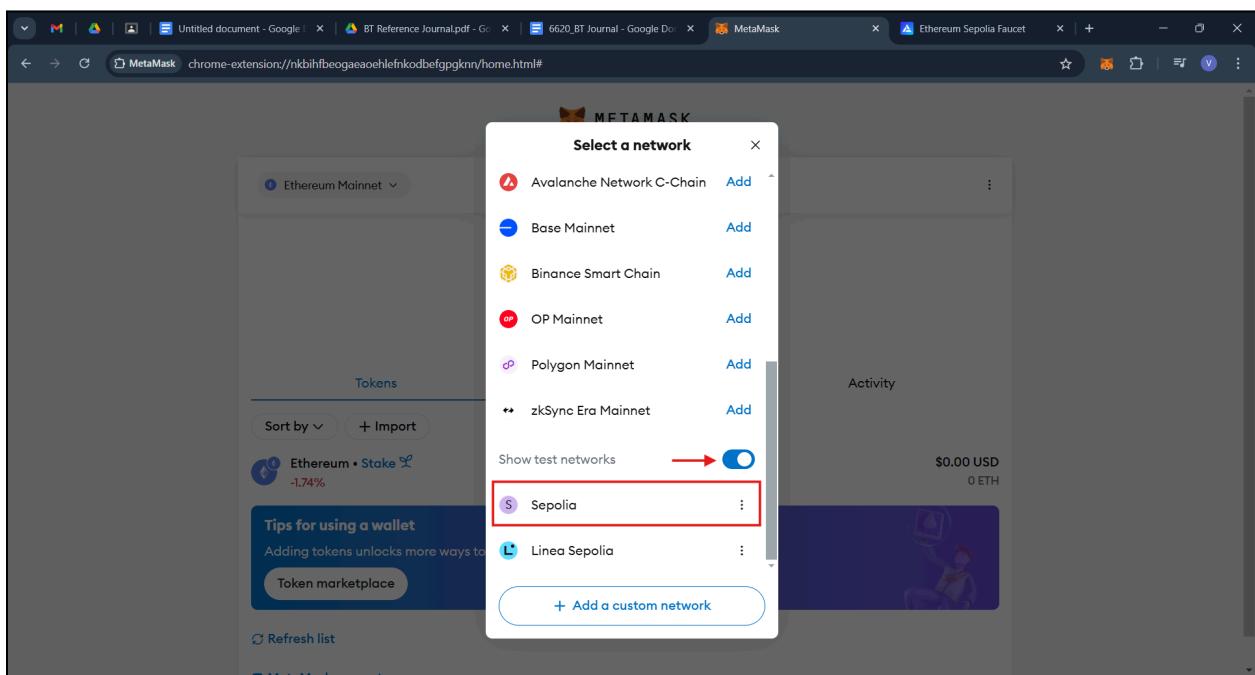
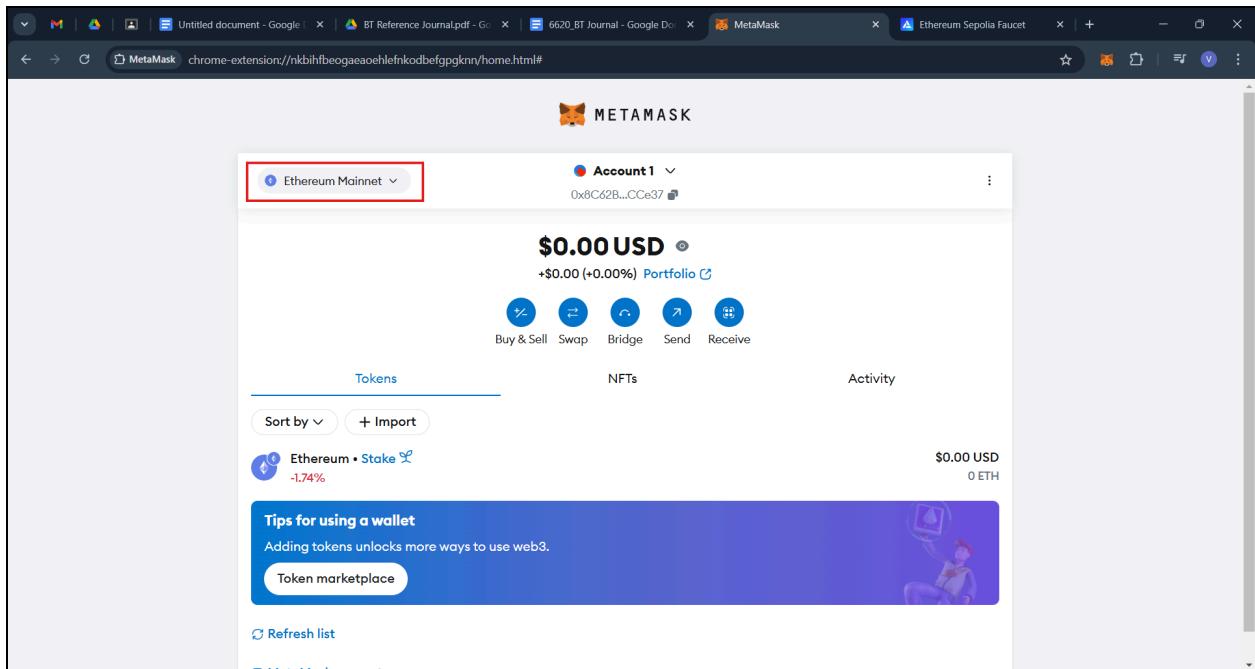
Create a wallet after adding an extension to chrome.

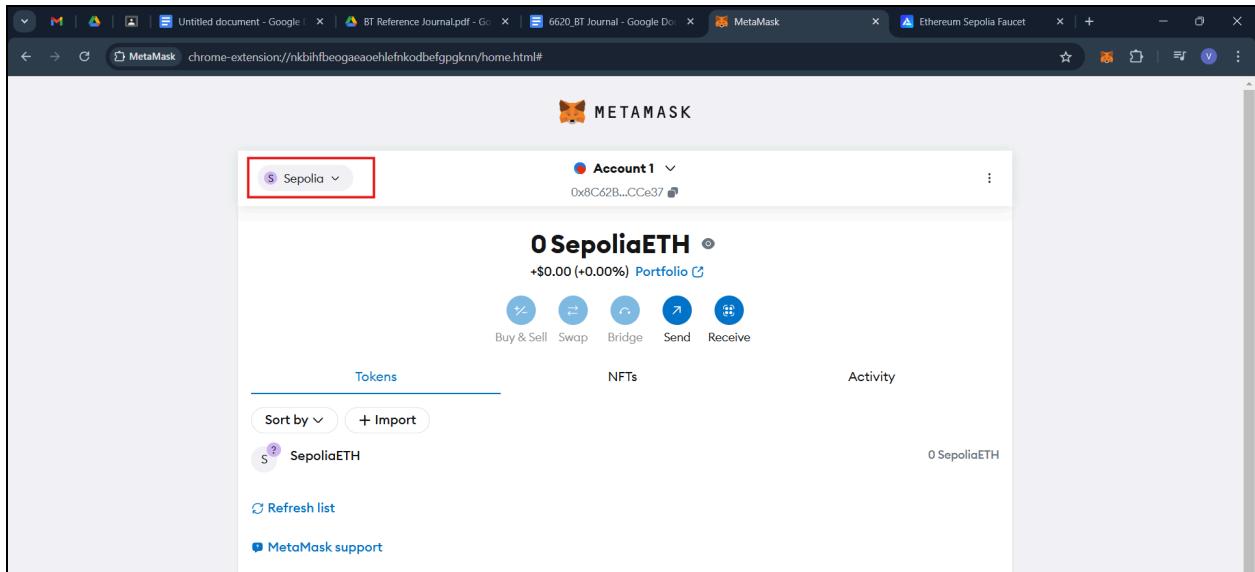


Set password and recovery key and your MetaMask is ready



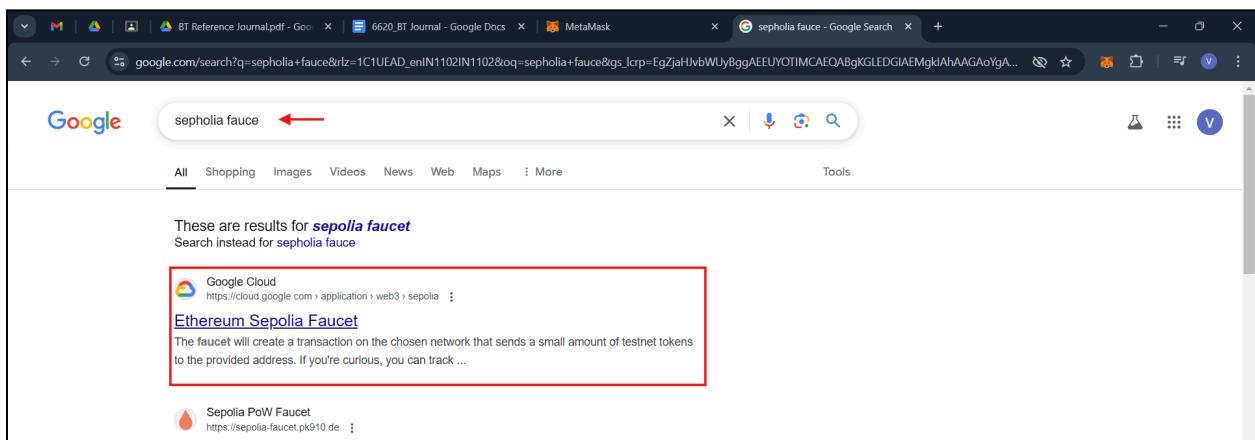
Now use Test Network Sepolia.



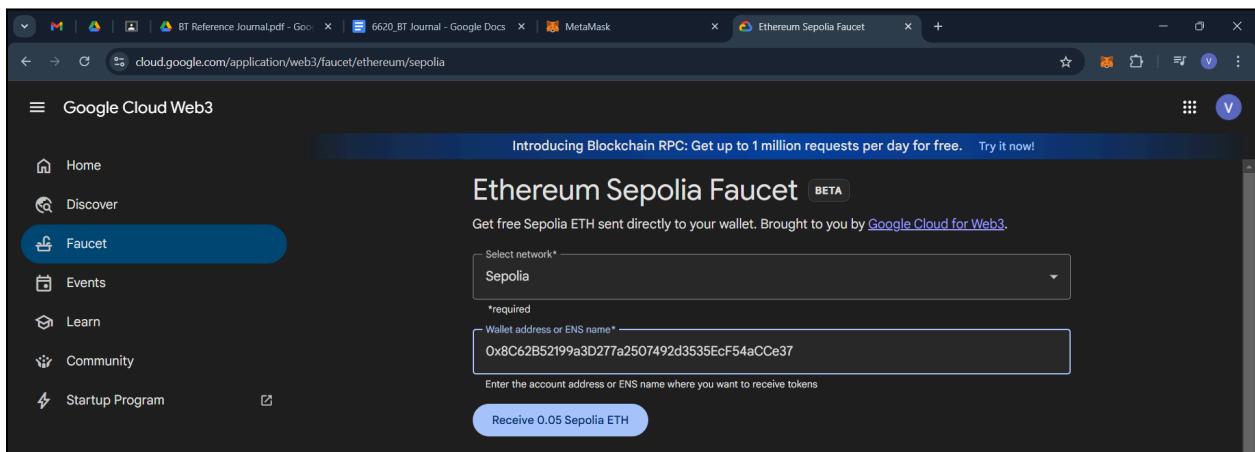


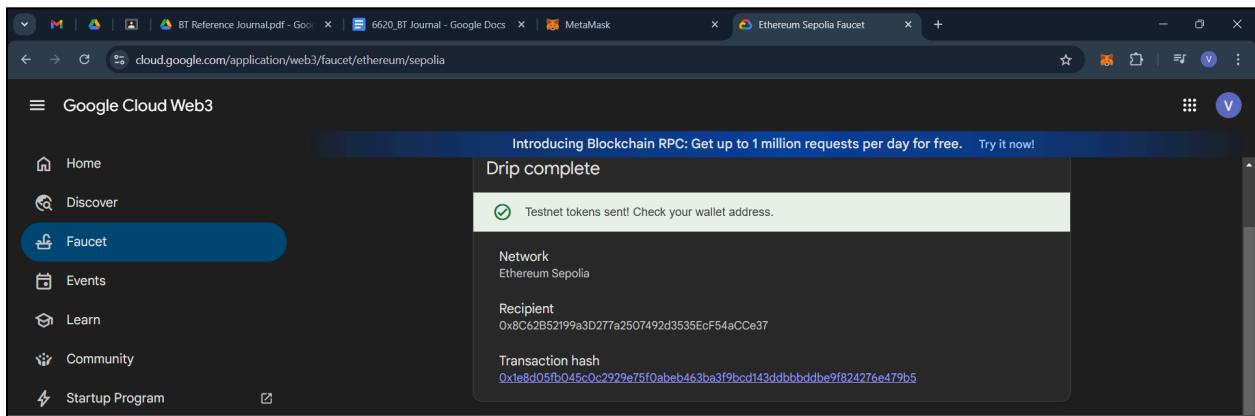
Add ETH into it.

Search Sepolia faucet on browser open google cloud link

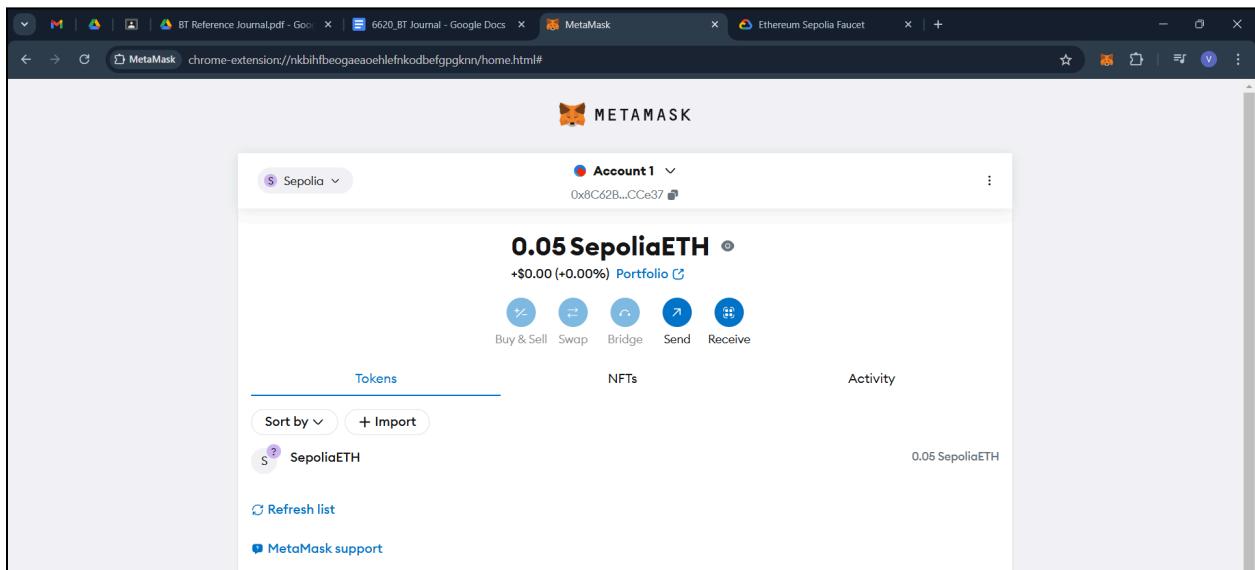


Enter your MetaMask Wallet address an click on receive ETH button





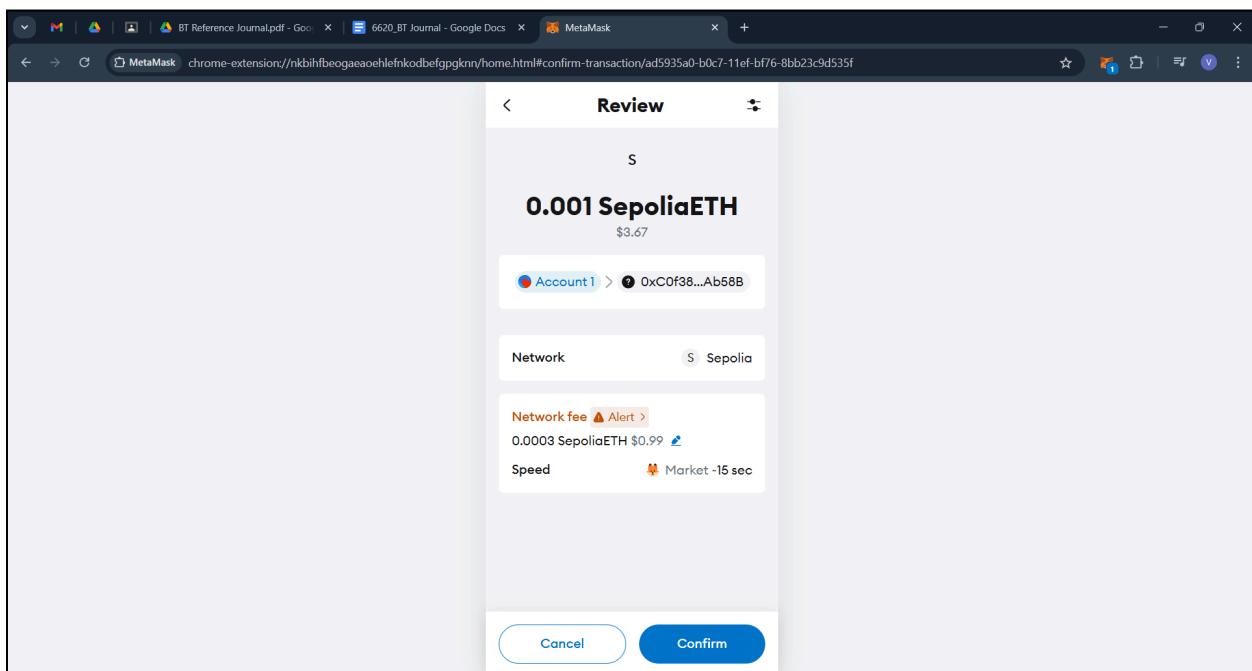
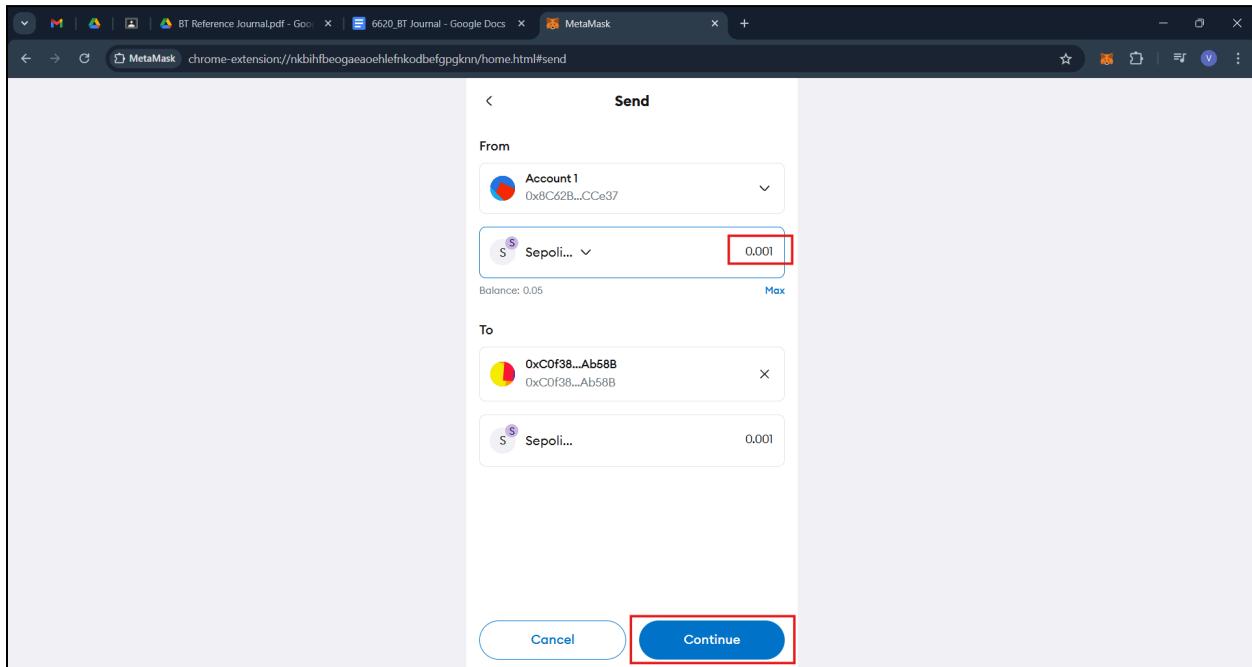
Check your wallet and 0.05 ETH are reflecting

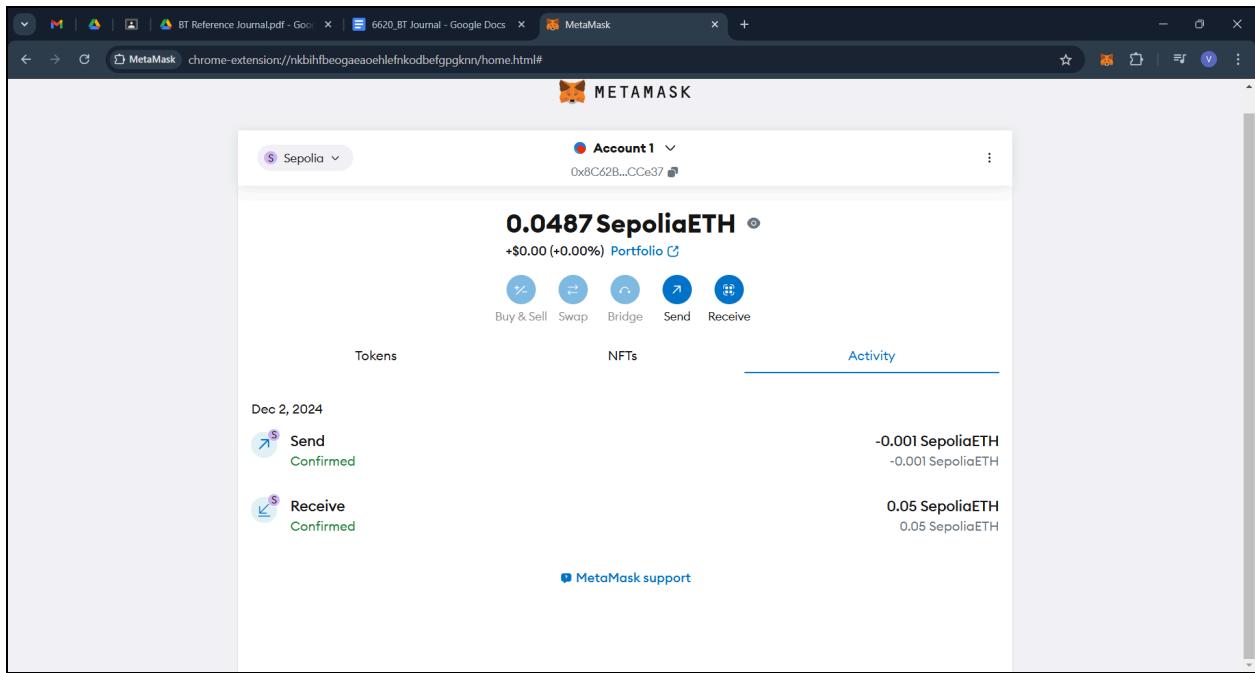


Once it is done now to see whether u can send or receive sepholia token or not now take your token and send it to you college and just sen 0.01 sepholia token with your friend

B. Transfer ether in between accounts and observe the transaction details

Click on send and add the address of the person you want to send





Practical 2: Implementing cryptography Algorithm in Python:

- a. DES
 - b. SHA Message Digest
-

A. DES

Data Encryption Standard (DES) is a symmetric key encryption algorithm that uses a block cipher to transform fixed-length blocks of plaintext into ciphertext. In Python, the Crypto library provides an implementation of DES. The `des_encrypt` function encrypts a given plaintext using DES with a specified key, and `des_decrypt` decrypts the ciphertext back to the original plaintext. DES, although historically significant, is considered insecure for modern applications, and more advanced algorithms like Advanced Encryption Standard (AES) are recommended.

Install the following library for further practicals

```
pip install pycryptodome
```

```
import datetime
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

def generate_key():
    return get_random_bytes(8)

def encrypt(message, key):
    cipher = DES.new(key, DES.MODE_ECB)
    encrypted_message = message.encode('utf-8')
    padded_message = pad(encrypted_message, DES.block_size)
    ciphertext = cipher.encrypt(padded_message)
    return ciphertext

def decrypt(ciphertext, key):
    cipher = DES.new(key, DES.MODE_ECB)
    decrypted_message = cipher.decrypt(ciphertext)
    unpadded_message = unpad(decrypted_message, DES.block_size)
    ciphertext = unpadded_message.decode('utf-8')
    return ciphertext
```

```
if __name__ == "__main__":
    print(datetime.datetime.now())
    key = generate_key()
    message = "6620_Shivam"
    ciphertext = encrypt(message, key)
    print("Encrypted:", ciphertext.hex())
    decrypted_message = decrypt(ciphertext, key)
    print("Decrypted:", decrypted_message)
```

```
PS D:\6620\BT> python -u "d:\6620\BT\des.py"
2024-12-03 07:59:32.780246
Encrypted: ba0e77db8fceca21e4e4c8f082275ac4
Decrypted: 6620_Shivam
PS D:\6620\BT>
```

Explanation:

generate_key(): Generates a random 8-byte key for DES encryption.

encrypt(message, key): Encrypts the given message using DES in ECB mode.

decrypt(ciphertext, key): Decrypts the ciphertext back to the original message.

pad() and unpad(): Ensures the message is padded to a multiple of DES block size (8 bytes).

B. SHA Message Digest

Secure Hash Algorithm (SHA) is a family of cryptographic hash functions designed to produce a fixed-size hash value, typically represented as a hexadecimal number. In Python, the built-in `hashlib` module provides implementations of various SHA algorithms. The `sha_digest` function demonstrates the SHA-256 algorithm, computing the hash digest of an input message. SHA is widely used for generating secure message digests in applications like data integrity verification and password hashing.

```
import datetime
import hashlib

def sha256(message):
    # Create a new SHA-256 hash object
    sha256_hash = hashlib.sha256()

    # Update the hash object with the bytes-like object (message)
    sha256_hash.update(message.encode('utf-8'))

    # Get the hexadecimal representation of the digest
    digest = sha256_hash.hexdigest()

    return digest

print(datetime.datetime.now())
message = "6620_Shivam"
result = sha256(message)
print(f"SHA-256 Digest for '{message}': {result}")
```

```
PS D:\6620\BT> python -u "d:\6620\BT\SHA_digest.py"
2024-12-03 08:02:08.175458
SHA-256 Digest for '6620_Shivam': 93e0ca924d001356b9131f9ce6f0824c579065620072add3efd4ba3808f5274c
PS D:\6620\BT> []
```

Practical 3: 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 .
-

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

RSA, named after its inventors Rivest, Shamir, and Adleman, is an asymmetric key encryption algorithm widely used for secure communication. It relies on the mathematical challenge of factoring large primes. Each participant has a pair of keys: a public key for encryption and a private key for decryption. RSA's security hinges on the difficulty of factoring large primes, making it a fundamental component in securing online transactions and communications.

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

class Client:
    def __init__(self):
        random_gen = Random.new().read
        self._private_key = RSA.generate(1024, random_gen)
        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.export_key(format='DER')).decode('ascii')

# Create a new client instance
client = Client()
print(datetime.datetime.now())
print("Sender: ", client.identity)
```

```
D:\MSc-IT\SEM 2\BT\Practical 2>python -u "d:\MSc-IT\SEM 2\BT\Practical 2\RSA.py"
2024-12-05 21:21:46.721930
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100ce3bfacdde6223b57bc9e77ccc5b422
261389344c746f6381197430915df517a44d135bc7e0e9b14355f916bc127b87e49551b01fde3d3934173fad1e1c25ebc2
760064475a872b9b6269eb1e977336233ad98678dfa6948ece11664fef2f5fb4d77955af50f5d09015e6daab366696e371
90ea7473fecfcb0ba5b5af6a044bb0203010001

D:\MSc-IT\SEM 2\BT\Practical 2>
```

Explanation:

Client class:

- Generates a 1024-bit RSA key pair (private and public keys).
- The public key is used to derive the client's identity (a unique identifier).

Identity property: Returns the hexadecimal representation of the public key.

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

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

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')

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('utf-8'))
    return binascii.hexlify(signer.sign(h)).decode('ascii')

def display_transaction(transaction):
    _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()

# Create two clients and a transaction between them
transactions = []
A = Client()
B = Client()

# Create a transaction from A to B
t1 = Transaction(A, B.identity, 15.0)

# Sign the transaction
signature = t1.sign_transaction()

# Display the transaction
display_transaction(t1)
print("Signature:", signature)

```

```
D:\MSc-IT\SEM 2\BT\Practical 2>python -u "d:\MSc-IT\SEM 2\BT\Practical 2\prac_3b.py"
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a4934eb6ffb3f0a873e209cb17d2e09e0
dbf6cccb1015954109c7be1cd47bd7185a4af3fb654fa7b6d2cf60bccbbd6e91773477fa518255695c2d7c0347471b2c4b1
dc054895af0c5dd6781257770ddb9fd9d7f3f8e9a2c3bdb9af4f0d85b5931517bc8ccb5af40b90d673eb0e56bad38c0f5b7
b01f2e1c76b5d14bc38e30a3f0203010001

recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100c3aff9c0e1c65c8914d5d33ee96802
ea97d8106cf2e844aa70c6d60f435bb74a83d75dec8427f1fe99b8de8d2a11ebcf708b05256243a0ffbcd08bbc99ac1
54fed8f9ee7ac5b3598e153392de4c4886d8b9c62c4c7ac628a2d6bb14341cca96d12318176fd78bc466f1ab5dc4234c82e
b7fb27895bb1187d7a7506f428630203010001

value: 15.0

time: 2024-12-05 21:27:46.391966

Signature: 97bf58a3c67e62f59aa94356b990fa17dea28ec0645c8c008bb7f0816fdb99bdbde7ee868b911ef677353a26
fa9824d8a9bdb0a56540be2e1a26496fc59b6b7c6db3b61b1f24f8c00c84c6ad7a36e3173e7376c826105746cb74a2b2581
dd402575a333270d28c07a36519b20d53d4db1e2be9e51fc741ba9d1bd00fc367f92

D:\MSc-IT\SEM 2\BT\Practical 2>
```

Explanation:

Client class:

- Generates an RSA key pair (private and public).
- The identity is the public key in hexadecimal format.

Transaction class:

- Represents a transaction with sender, recipient, and value.
- `to_dict()`: Converts the transaction details into an ordered dictionary.
- `sign_transaction()`: Signs the transaction using the sender's private key.

`display_transaction()`: Prints the details of the transaction.

Practical 4: Write the following programs for Blockchain in Python :

- a. Create multiple transactions and display them.
 - b. Create a blockchain, a genesis block and execute it .
-

A. Create multiple transactions and display them.

Multiple transactions within a blockchain refer to the numerous data exchanges that occur between participants in a blockchain network. Each transaction involves the transfer or modification of digital assets or information, and these transactions are grouped together into blocks. The blockchain, as a decentralized and distributed ledger, ensures transparency and security by recording these transactions in a sequential and immutable manner across all network nodes.

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

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 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('utf-8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')

def display_transaction(transaction):
    _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()

# Create clients
alice = Client()
bob = Client()
jhon = Client()

# Create transactions
transactions = []

```

```

t1 = Transaction(alice, bob.identity, 15.0)
t1.sign_transaction()
transactions.append(t1)

t2 = Transaction(bob, jhon.identity, 25.0)
t2.sign_transaction()
transactions.append(t2)

t3 = Transaction(bob, jhon.identity, 200.0)
t3.sign_transaction()
transactions.append(t3)

# Display all transactions
tn = 1
for t in transactions:
    print("Transaction #", tn)
    display_transaction(t)
    tn += 1
print()

```

```

D:\MSc-IT\SEM 2\BT\Practical 2>python -u "d:\MSc-IT\SEM 2\BT\Practical 2\prac_4a.py"
Transaction # 1
sender: 30819f300d06092a864886f70d010101050003818d00308189028181008d440c99df1d46b7625150f6c8952b12
929b063ac12a78d21c27c2b257194cf35b09cf8cf1f92d25695e78510ff0997a60ad940ad5ab2c8d915609bf8005d313b7e
82a78b75b90d847ede2a15f460855a5a489f37b48d8bee2e1858b1a57a75fb381502745f0d397c0928ff5c7bcd6bc78b5
7d94b2ae8637285ff60f0678d0203010001

recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b7c03845b1c302f267a965cd8311fe
e00ea681694aee2ad9ce4ae392139f068f702509d4e504392bdbe73356ff4686686581e012b402943ec2b422e93c60ebcb1
2d341cde23c3b5566a3a2c3752cfc846c40fd8ce1499fccd5f3ae03b5e80a0297b2bba4d115a14d200f8bedb5905965ab5
d50d2a48037a8c2ce38c1c6d2b390203010001

value: 15.0

time: 2024-12-05 21:32:10.064159

```

```

Transaction # 2
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100b7c03845b1c302f267a965cd8311fee00
ea681694aee2ad9ce4ae392139f068f702509d4e504392bdbe73356ff4686686581e012b402943ec2b422e93c60ebcb12d3
41cde23c3b5566a3a2c3752cfc846c40fd8ce1499fccd5f3ae03b5e80a0297b2bba4d115a14d200f8bedb5905965ab5d0
d2a48037a8c2ce38c1c6d2b390203010001

recipient: 30819f300d06092a864886f70d010101050003818d00308189028181009e7077bb1ae152eef67359cfe47efb
b526964266021aae57ad382562eea8247150837cd5a49ef4d24f10424e276dfe6356671daf430070232bcc9d58755c71bf0
f8e9b30a69a296360242c159c8a287dcec8ea63013946f4bd5d5bc68a0fb6285bcc59c6c804752171520266b28c0dca7cc0
1c8ecfa7cb4a97792cd1da6a43490203010001

value: 25.0

time: 2024-12-05 21:32:10.064159

```

```
Transaction # 3
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100b7c03845b1c302f267a965cd8311fee00
ea681694aee2ad9ce4ae392139f068f702509d4e504392bdb73356ff4686686581e012b402943ec2b422e93c60ebcb12d3
41cde23c3b5566a3a2c3752cfca846c40fd8ce1499fccd5f3ae03b5e80a0297b2bba4d115a14d200f8bedb5905965ab5d50
d2a48037a8c2ce38c1c6d2b390203010001

recipient: 30819f300d06092a864886f70d010101050003818d00308189028181009e7077bb1ae152eef67359cfe47efb
b526964266021aae57ad382562eea8247150837cd5a49ef4d24f10424e276dfe6356671daf430070232bcc9d58755c71bf0
f8e9b30a69a296360242c159c8a287dcec8ea63013946f4bd5d5bc68a0fb6285bcc59c6c804752171520266b28c0dca7cc0
1c8ecfa7cb4a97792cd1da6a43490203010001

value: 200.0

time: 2024-12-05 21:32:10.064159
```

Explanation:

Client class:

- Generates a 1024-bit RSA key pair.
- The identity property returns the public key in hexadecimal form.

Transaction class:

- Represents a transaction with a sender, recipient, and value.
- `to_dict()`: Converts the transaction into an ordered dictionary.
- `sign_transaction()`: Signs the transaction using the sender's private key.

`display_transaction()`: Prints the details of each transaction.

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

The term "genesis block" refers to the inaugural block in a blockchain. It serves as the starting point of the entire blockchain network and contains no reference to a previous block. The creation of the genesis block marks the initiation of the blockchain, and subsequent blocks build upon it as the network grows. The genesis block often includes certain unique characteristics or messages, and its details are crucial for verifying the integrity of the entire blockchain.

```
import hashlib
import datetime

class Block:
    def __init__(self, index, previous_hash, timestamp, data, hash):
        self.index = index
        self.previous_hash = previous_hash
        self.timestamp = timestamp
        self.data = data
        self.hash = hash

    def calculate_hash(index, previous_hash, timestamp, data):
        block_info = f"{index}{previous_hash}{timestamp}{data}"
        return hashlib.sha256(block_info.encode()).hexdigest()

    def create_block(index, previous_hash, data):
        timestamp = datetime.datetime.now()
        hash = calculate_hash(index, previous_hash, timestamp, data)
        return Block(index, previous_hash, timestamp, data, hash)

# Create a blockchain with a genesis block
blockchain = [Block(0, "0" * 64, datetime.datetime.now(), "Genesis Block",
                    calculate_hash(0, "0" * 64, datetime.datetime.now(),
"Genesis Block"))]

# Add a new block to the blockchain
data_for_new_block = "Some data for the new block"
new_block = create_block(len(blockchain), blockchain[-1].hash,
data_for_new_block)
blockchain.append(new_block)
```

```
# Display the state of the blockchain
for block in blockchain:
    print(f"\nBlock # {block.index} - Timestamp: {block.timestamp}, Data: {block.data}, Hash: {block.hash}")
```

```
D:\MSc-IT\SEM 2\BT\Practical 2>python -u "d:\MSc-IT\SEM 2\BT\Practical 2\prac_4b.py"

Block #0 - Timestamp: 2024-12-05 21:34:52.369565, Data: Genesis Block, Hash: 38f2f144b7505fb9e5e22aa5c0e0f32f28ed7c8653e799fc240c366d99264774

Block #1 - Timestamp: 2024-12-05 21:34:52.369565, Data: Some data for the new block, Hash: 7afce96579d46c200ec864a699c2338c268617431a8cbc4587e423c427cad0e

D:\MSc-IT\SEM 2\BT\Practical 2>[REDACTED]
```

Explanation:

Block class: Represents a block in the blockchain with the following properties: index, previous_hash, timestamp, data, and hash.

calculate_hash function: Computes the SHA-256 hash of a block based on its index, previous_hash, timestamp, and data.

create_block function: Creates a new block with a unique hash.

Genesis block: The first block in the blockchain, which has a fixed hash of 0 for the previous_hash.

Adding a new block: A new block is added to the blockchain after the genesis block, with some sample data.

Display blockchain: Prints out each block's index, timestamp, data, and hash.

Practical 5: Write the following programs for Blockchain in Python :

- Create a mining function and test it.
-

- Create a mining function and test it.

In a blockchain network, mining is the process by which new blocks are added to the blockchain through the completion of complex mathematical computations. The primary purpose of mining is to secure and validate transactions within the network, ensuring the integrity of the decentralized ledger.

```
import hashlib

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

def mine(message, difficulty=1):
    assert difficulty >= 1
    # Prefix of '1's based on difficulty
    prefix = '1' * difficulty
    print("Prefix:", prefix)

    for i in range(1000):
        # Create a hash using the message and the nonce (i)
        digest = sha256(str(hash(message)) + str(i))
        print("Testing --> " + digest)

        # Check if the hash starts with the desired prefix
        if digest.startswith(prefix):
            print("\nAfter", i, "iterations,\nfound nonce:", digest)
            return i    # Return the nonce value

# Run mining with difficulty level 2
mine("test message", 2)
```

```
D:\MSc-IT\SEM 2\BT\Practical 2>python -u "d:\MSc-IT\SEM 2\BT\Practical 2\prac_5a.py"
Prefix: 11
Testing --> ce21913faf1526a80694692a491fa8a5b47ec65cbd9ff3f2e2f5a79bd4db0aa6
Testing --> 0e12baf757c3b87babf966a9ba12a82649c5628d6707daf9c31450f7ef43f55a
Testing --> 7d3ab115d1a78bcda8ce0aeb9ed77ad5bc3fdc3183aea154c0c3d97cd723bcc
Testing --> 6b0442e5d00348da054feb84ae9d203a023d2098bf787a35dd0b436e9e63001d
Testing --> 236cbe66095bdece65cd73d8d48b108477784e9bc14a0ce43a520d11a448c9d7
Testing --> 0dddc47ca4c4b6a43e45bf020119c7d5e83098bcc6a7f8bed78af297db1a401f
```

```
Testing --> 3c7fd3fcf34c6683e19661f41a3b77119c8d261d2855f20d94c3c2f4c852563a
Testing --> bd3d0f38247fb9217b09f5829fe414c53645ebb3bf91597b5da8757728b6eaad
Testing --> eab00e1d9fc4792bd3a3871998e6a03e5e54803c8b25d492aa6c8ca1be4b7303
Testing --> 11597f5efcf4ba7a9074fac0e08dde0ee94c345318f4d614898a87934ee1d43
```

```
After 227 iterations,
found nonce: 11597f5efcf4ba7a9074fac0e08dde0ee94c345318f4d614898a87934ee1d43
```

```
D:\MSc-IT\SEM 2\BT\Practical 2>
```

Explanation:

sha256 function: Takes a message, encodes it into ASCII, and returns the SHA-256 hash of the message.

mine function:

- Attempts to find a nonce (a number) that, when appended to the message and hashed, results in a digest that starts with a specific number of '1's, determined by the difficulty parameter.
- The function runs for up to 1000 iterations, printing each hash it checks.
- If it finds a hash that starts with the specified number of '1's, it prints the nonce and the hash and returns the nonce value.

Prefix: The prefix is a string consisting of '1' repeated for the number of times defined by the difficulty.

Additional Notes:

- The difficulty parameter determines how many '1's the hash needs to start with (e.g., difficulty 1 means the hash must start with '1', difficulty 2 means it must start with '11').
- The hash will be tested up to 1000 iterations, and the first valid nonce will be returned.

Practical 6: Write the solidity Code and demonstrate the following:

- a. Variable
 - b. Operators
 - c. Decision Making - if, if else, if else if
 - d. Loops - for, while, and do while
-

A. Variable

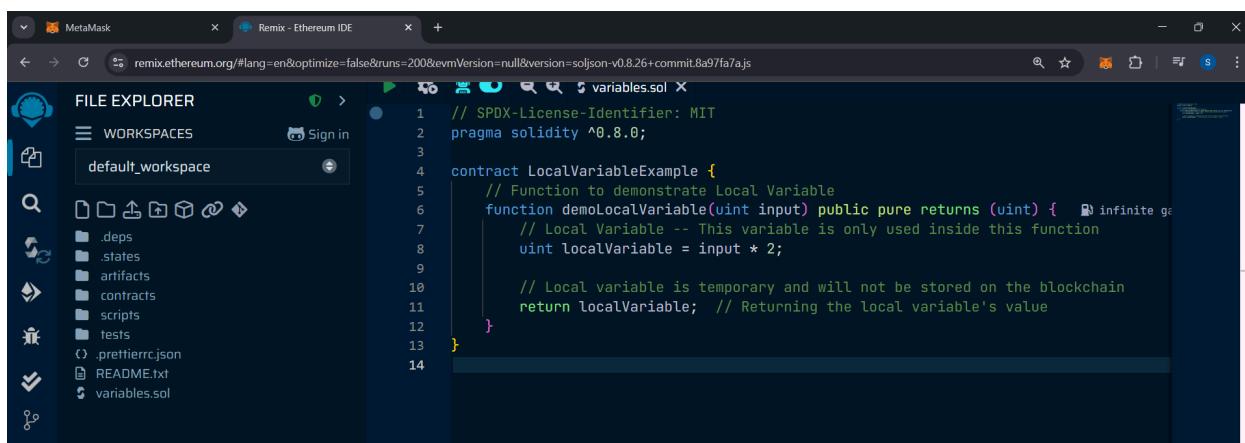
1. Local Variable

Local variables are declared within a specific function or code block and exist only for the duration of that particular execution. They are used for temporary storage and calculations within the confines of a single function.

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

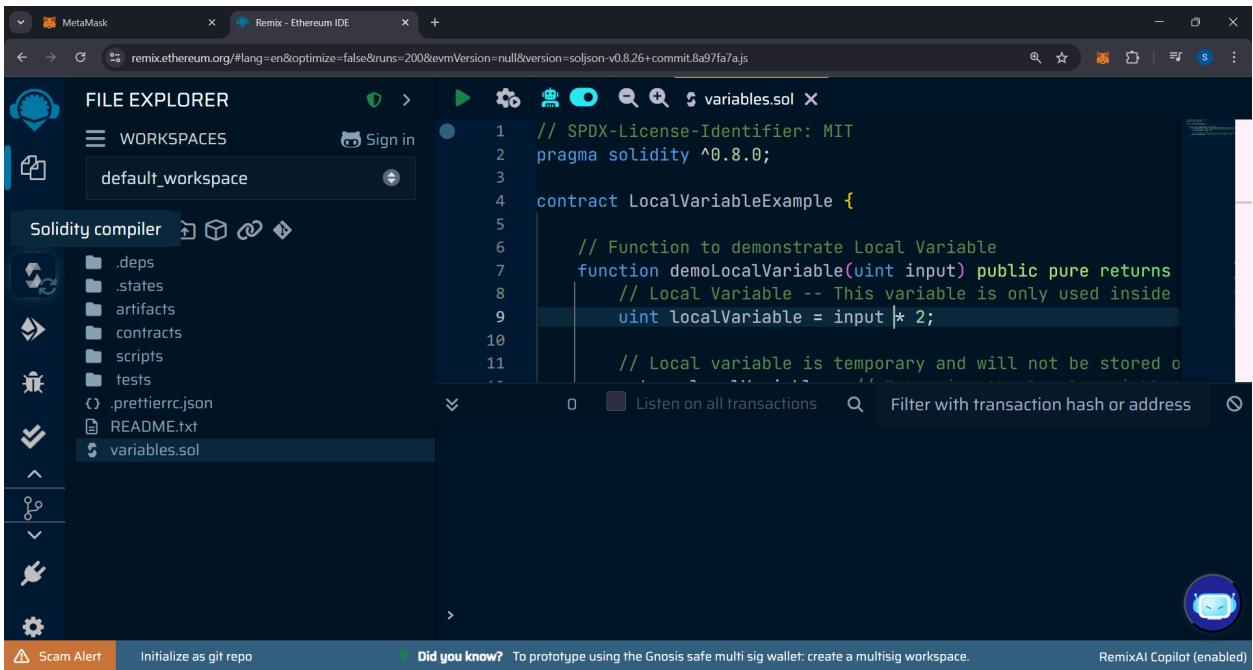
contract LocalVariableExample {
    // Function to demonstrate Local Variable
    function demoLocalVariable(uint input) public pure returns (uint) {
        // Local Variable -- This variable is only used inside this function
        uint localVariable = input * 2;

        // Local variable is temporary and will not be stored on the blockchain
        return localVariable; // Returning the local variable's value
    }
}
```



Steps to run

Go to Compiler from Icon menu



The screenshot shows the Remix Ethereum IDE interface. On the left is the FILE EXPLORER panel, which displays the workspace structure: default_workspace, .deps, states, artifacts, contracts, scripts, tests, .prettierrc.json, README.txt, and variables.sol. The main central area is the Solidity compiler, showing the Solidity code for a contract named LocalVariableExample. The code includes comments explaining local variables and their scope. At the bottom of the code editor, there are buttons for Listen on all transactions and Filter with transaction hash or address. A sidebar on the right contains various icons for interacting with the blockchain.

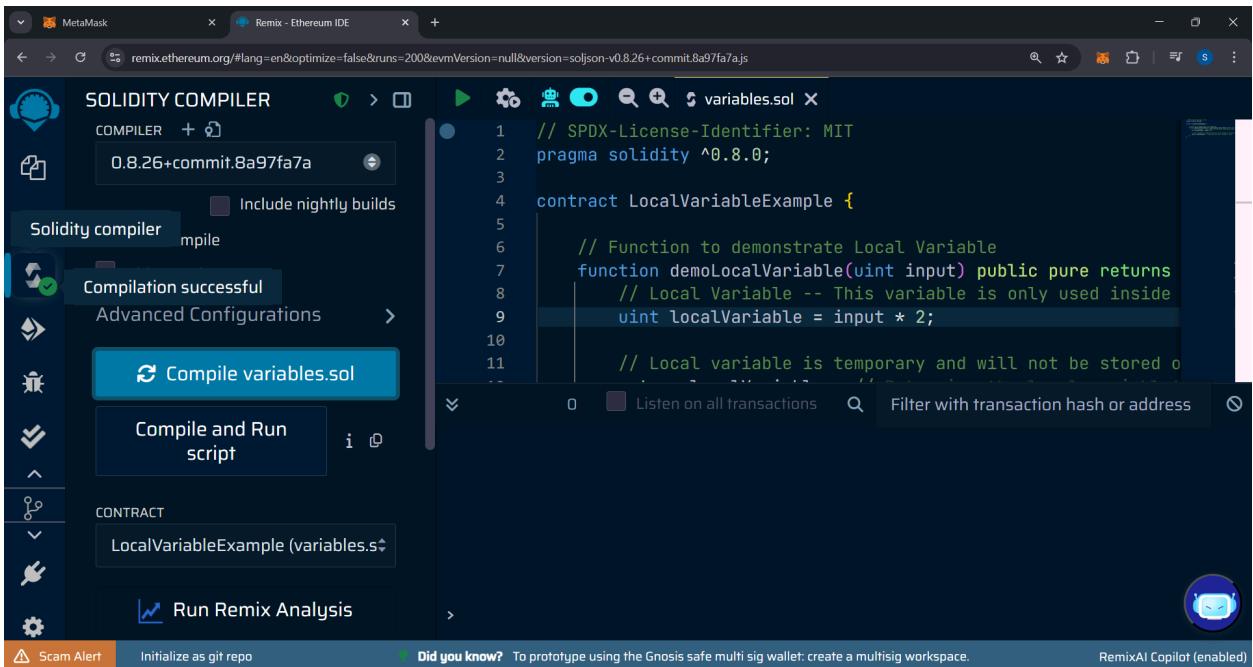
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract LocalVariableExample {

    // Function to demonstrate Local Variable
    function demoLocalVariable(uint input) public pure returns
        // Local Variable -- This variable is only used inside
        uint localVariable = input * 2;

    // Local variable is temporary and will not be stored o
}
```

Click on Solidity Compile button



The screenshot shows the Remix Ethereum IDE interface after compilation. The Solidity compiler sidebar now displays "Compilation successful". Below it, there are buttons for "Compile variables.sol" (which is highlighted in blue), "Compile and Run script", and "Contract" (which lists "LocalVariableExample (variables.sol)"). At the bottom of the code editor, there are buttons for "Run Remix Analysis" and "RemixAI Copilot (enabled)". The rest of the interface remains the same as the previous screenshot.

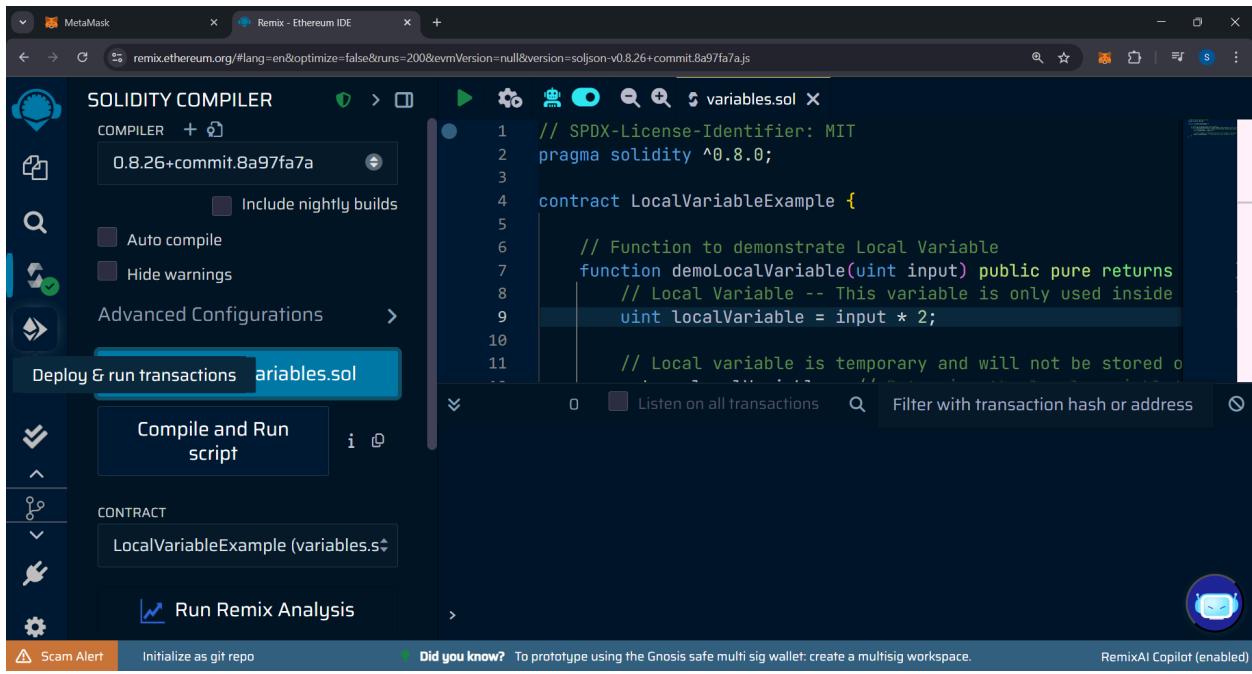
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract LocalVariableExample {

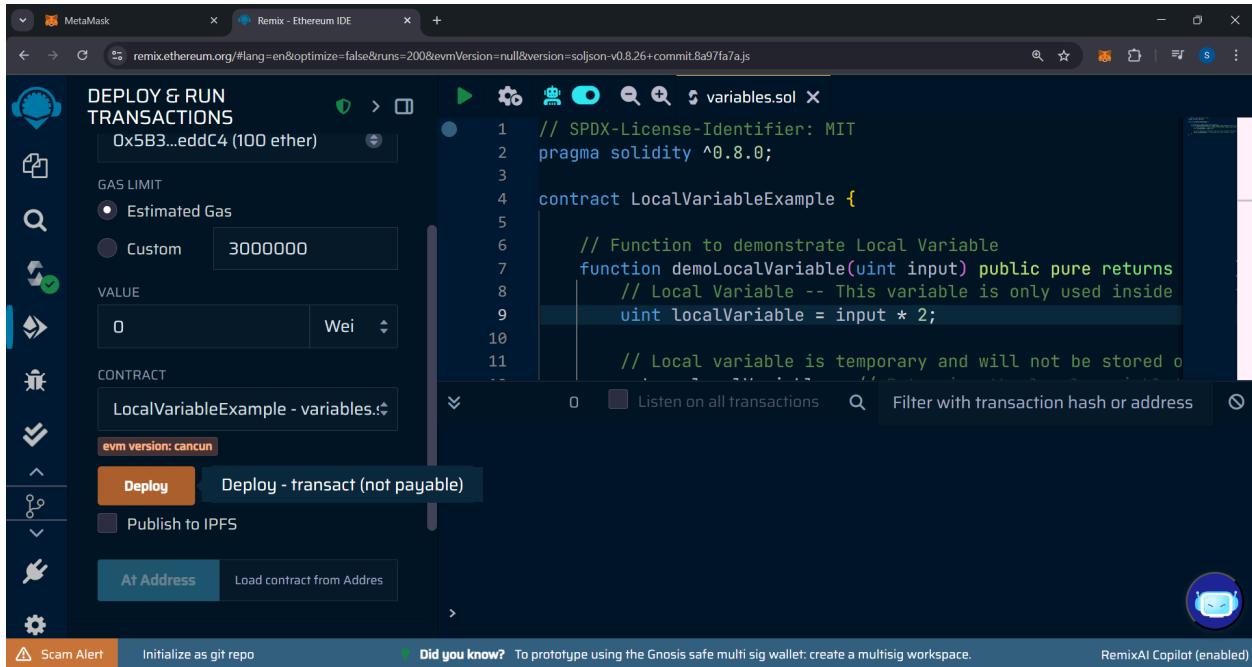
    // Function to demonstrate Local Variable
    function demoLocalVariable(uint input) public pure returns
        // Local Variable -- This variable is only used inside
        uint localVariable = input * 2;

    // Local variable is temporary and will not be stored o
}
```

Go to deploy and run transaction tab and click on deploy button



The screenshot shows the Remix Ethereum IDE interface. On the left, there's a sidebar with various icons for file operations like Open, Save, and Import. The main area is divided into sections: 'SOLIDITY COMPILER' on the top left, a code editor on the right containing a Solidity contract named 'variables.sol', and a 'CONTRACT' section below it. In the 'CONTRACT' section, the contract name 'LocalVariableExample (variables.sol)' is selected. At the bottom of the interface, there are several status bars and notifications, including 'Scam Alert', 'Initialize as git repo', 'Did you know?', and 'RemixAI Copilot (enabled)'. The 'Deploy & run transactions' button is highlighted with a blue background.



This screenshot shows the same Remix Ethereum IDE interface, but the 'DEPLOY & RUN TRANSACTIONS' tab is now active. The 'GAS LIMIT' section shows '0x5B3...eddC4 (100 ether)' and 'Estimated Gas' selected. The 'VALUE' section shows '0 Wei'. The 'CONTRACT' section remains the same. At the bottom, the 'Deploy' button is highlighted with a blue background. Other buttons like 'Deploy - transact (not payable)' and 'Publish to IPFS' are also visible.

Your contract will be deployed just scroll down and will can see you contract name

The screenshot shows the Remix Ethereum IDE interface. On the left, the sidebar displays "DEPLOY & RUN TRANSACTIONS" with a "Deploy" button and a "Deployed Contracts" section containing "LOCALVARIABLEEXAMPLE AT". The main area shows the Solidity code for "variables.sol":

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

contract LocalVariableExample {
    // Function to demonstrate Local Variable
    function demoLocalVariable(uint input) public pure returns
        // Local Variable -- This variable is only used inside
        uint localVariable = input * 2;
    }

    // Local variable is temporary and will not be stored o
```

The status bar at the bottom indicates "creation of LocalVariableExample pending...". A transaction log is shown in the bottom right:

[vm] from: 0x5B3...eddC4 to: LocalVariableExample.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0x548...08622

Enter input number and check the output

The screenshot shows the Remix Ethereum IDE interface after deployment. The "Deployed Contracts" section now includes "LOCALVARIABLEEXAMPLE AT". The "demoLocalVariable" section on the left has an "input" field set to "6620". A "call" button is highlighted. The status bar at the bottom indicates "call to LocalVariableExample.demoLocalVariable". A transaction log is shown in the bottom right:

[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: LocalVariableExample.demoLocalVariable(uint256) data: 0x559...019dc

2. State Variable

State variables, on the other hand, are declared outside any function within the contract and have a more extended lifetime. They persist across multiple function calls and endure as long as the contract is deployed.

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

contract StateVariableExample {

    // State Variable -- This variable is stored on the blockchain
    uint public stateVar;

    // Function to demonstrate updating State Variable
    function setStateVariable(uint newValue) public {
        stateVar = newValue; // Assigning new value to state variable
    }
}
```

Output

The screenshot shows the Remix Ethereum IDE interface. On the left, the sidebar displays 'Transactions recorded' (6), 'Deployed Contracts' (1), and the specific contract 'STATEVARIABLEEXAMPLE A'. The main area shows the Solidity code for 'variables.sol'. Below the code, the 'stateVar' variable is shown with its current value as 0. A transaction input field is present, with 'setStateVariable' selected and 'newValue' set to 0. At the bottom, there are buttons for 'Transact' and 'Deploy'.

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

contract StateVariableExample {

    // State Variable -- This variable is stored on the blockchain
    uint public stateVar;

    // Function to demonstrate updating State Variable
    function setStateVariable(uint newValue) public {
        stateVar = newValue; // Assigning new value to state variable
    }
}
```

The screenshot shows the Remix Ethereum IDE interface. On the left, there's a sidebar with icons for MetaMask, Deploy & Run Transactions (with options to publish to IPFS or run at an address), Transactions recorded (6), Deployed Contracts (1: STATEVARIABLEEXAMPLE A), Balance: 0 ETH, and a setStateVariable section. In the center, the Solidity code for `variables.sol` is displayed:

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

contract StateVariableExample {
    // State Variable -- This variable is stored on the blockchain
    uint public stateVar;

    // Function to demonstrate updating State Variable
    function setStateVariable(uint newValue) public {
        stateVar = newValue; // Assigning new value to state variable
    }
}
```

In the `setStateVariable` section, the `newValue` field is set to "6620". Below it, the `Calldata` and `Parameters` fields are shown, with the `Parameters` button highlighted in orange. The text "setStateVariable - transact (not payable)" is displayed above the parameters. At the bottom of the sidebar, there are sections for `Low level interactions` and `CALDATA`. The footer of the IDE includes a Scam Alert icon, a link to Initialize as git repo, a Did you know? tip about Gnosis safe multisig wallets, and a RemixAI Copilot status indicator.

This screenshot is nearly identical to the first one, showing the Remix Ethereum IDE interface. The Solidity code and transaction setup are the same. However, the `setStateVariable` section now shows the result of the transaction: the `stateVar` value is listed as "0: uint256: 6620". The rest of the interface, including the sidebar, code editor, and footer, remains the same.

3. Global Variable

Function parameters are variables declared within a function's signature, serving as inputs to the function. Similar to local variables, function parameters have a limited scope and lifetime, existing only for the duration of the function's execution. They allow external values to be passed into a function, enabling it to perform computations or modifications based on the provided inputs.

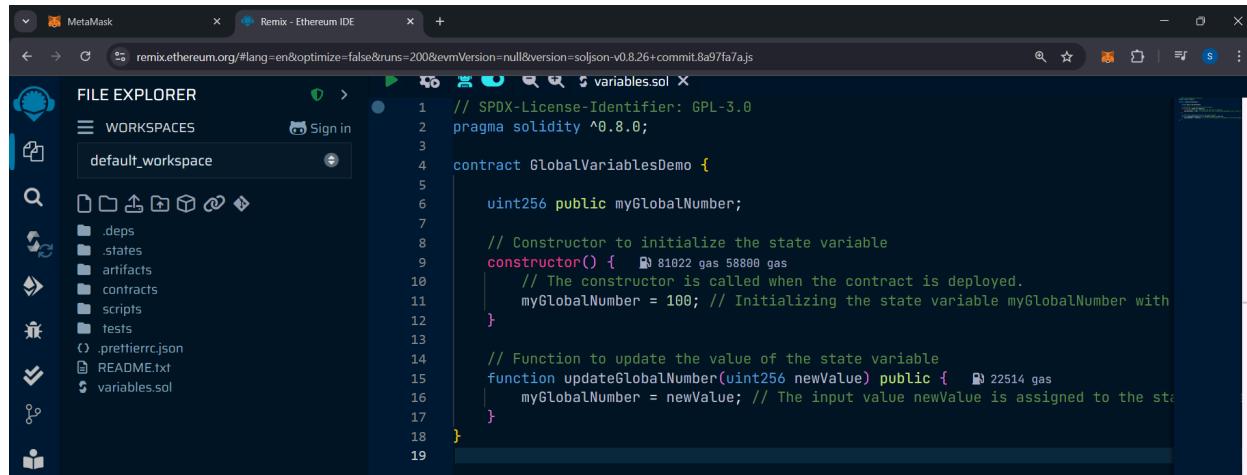
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract GlobalVariablesDemo {

    uint256 public myGlobalNumber;

    // Constructor to initialize the state variable
    constructor() {
        // The constructor is called when the contract is deployed.
        myGlobalNumber = 100; // Initializing the state variable myGlobalNumber with a value
        of 0.
    }

    // Function to update the value of the state variable
    function updateGlobalNumber(uint256 newValue) public {
        myGlobalNumber = newValue; // The input value newValue is assigned to the state
        variable myGlobalNumber.
    }
}
```



Output

The image displays two side-by-side screenshots of the Truffle UI interface, showing the state of a deployed contract named `GLOBALVARIABLESDEMO`.

Left Screenshot (Initial State):

- Deployed Contracts:** 1
- GLOBALVARIABLESDEMO**:
 - Balance:** 0 ETH
 - updateGlobalNumber**: `uint256 newValue`
 - myGlobalNumber - call**:
 - `0: uint256: 100`
 - Low level interactions**:
 - CALldata**:
- Scam Alert**
- Initialize as git repo**

Right Screenshot (After Transaction):

- Deployed Contracts:** 1
- GLOBALVARIABLESDEMO**:
 - Balance:** 0 ETH
 - updateGlobalNumber**:
 - newValue:** "6620"
 - Calldata**
 - Parameters**
 - transact** (highlighted in orange)
 - myGlobalNumber - call**:
 - `0: uint256: 100`
 - Low level interactions**:
 - CALldata**
- Scam Alert**
- Initialize as git repo**

This screenshot shows the state of the `GLOBALVARIABLESDEMO` contract after the `updateGlobalNumber` transaction has been executed.

- Deployed Contracts:** 1
- GLOBALVARIABLESDEMO**:
 - Balance:** 0 ETH
 - updateGlobalNumber**:
 - newValue:** "6620"
 - Calldata**
 - Parameters**
 - transact** (highlighted in orange)
 - myGlobalNumber - call**:
 - `0: uint256: 6620`
 - Low level interactions**:
 - CALldata**:
- Scam Alert**
- Initialize as git repo**

B. Operators

Solidity, Ethereum's smart contract language,

- features operators for arithmetic (e.g., +, -, *, /),
- comparisons (==, !=, <, >),
- logical (&&, ||, !),
- bitwise (&, |, ^, ~, <<, >>),
- assignment (=, +=, -=).

These operators empower developers to perform various computations, comparisons, and bit-level operations efficiently within smart contracts.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract Operators {
    // Arithmetic operators
    uint public additionResult = 5 + 3;          // 5 + 3 = 8
    uint public subtractionResult = 10 - 4;        // 10 - 4 = 6
    uint public multiplicationResult = 6 * 2;      // 6 * 2 = 12
    uint public divisionResult = 16 / 2;           // 16 / 2 = 8
    uint public moduloResult = 15 % 4;             // 15 % 4 = 3

    // Comparison operators
    bool public isEqual = (7 == 7);                // true, since 7 is equal to 7
    bool public isNotEqual = (5 != 3);              // true, since 5 is not equal to 3
    bool public isGreaterThan = (10 > 5);          // true, since 10 is greater than 5
    bool public isLessThan = (3 < 8);               // true, since 3 is less than 8
    bool public isGreaterOrEqual = (6 >= 6);        // true, since 6 is equal to 6
    bool public isLessOrEqual = (9 <= 10);          // true, since 9 is less than 10

    // Logical operators
    bool public andOperator = true && false;      // false, since both must be true for '&&' to
    return true
    bool public orOperator = true || false;          // true, since one is true for '||' to return true
    bool public notOperator = !true;                 // false, since !true is false
}
```

The screenshot shows the Remix Ethereum IDE interface. On the left, the FILE EXPLORER sidebar displays the project structure, including a workspace named "default_workspace" containing files like ".deps", ".states", "artifacts", "contracts", "scripts", "tests", ".prettierrc.json", "operators.sol", and "README.txt". The main editor area shows the Solidity code for the "operators.sol" contract. The code defines various arithmetic, comparison, and logical operators with comments explaining their results.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract Operators {
    // Arithmetic operators
    uint public additionResult = 5 + 3;           // 5 + 3 = 8
    uint public subtractionResult = 10 - 4;         // 10 - 4 = 6
    uint public multiplicationResult = 6 * 2;       // 6 * 2 = 12
    uint public divisionResult = 16 / 2;           // 16 / 2 = 8
    uint public moduloResult = 15 % 4;             // 15 % 4 = 3

    // Comparison operators
    bool public isEqual = (7 == 7);                // true, since 7 is equal to 7
    bool public isNotEqual = (5 != 3);              // true, since 5 is not equal to 3
    bool public isGreaterThan = (10 > 5);          // true, since 10 is greater than 5
    bool public isLessThan = (3 < 8);               // true, since 3 is less than 8
    bool public isGreaterOrEqual = (6 >= 6);        // true, since 6 is equal to 6
    bool public isLessOrEqual = (9 <= 10);          // true, since 9 is less than 10

    // Logical operators
    bool public andOperator = true && false;      // false, since both must be true for '&&' to
    bool public orOperator = true || false;          // true, since one is true for '||' to
    bool public notOperator = !true;                 // false, since !true is false
}
```

Output

The screenshot shows the Remix Ethereum IDE interface with the "DEPLOY & RUN TRANSACTIONS" sidebar open. The sidebar lists various functions and operators that can be interacted with. The main editor area shows the same Solidity code for the "operators.sol" contract, which includes comments explaining the results of different operations.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract Operators {
    // Arithmetic operators
    uint public additionResult = 5 + 3;           // 5 + 3 = 8
    uint public subtractionResult = 10 - 4;         // 10 - 4 = 6
    uint public multiplicationResult = 6 * 2;       // 6 * 2 = 12
    uint public divisionResult = 16 / 2;           // 16 / 2 = 8
    uint public moduloResult = 15 % 4;             // 15 % 4 = 3

    // Comparison operators
    bool public isEqual = (7 == 7);                // true, since 7 is equal to 7
    bool public isNotEqual = (5 != 3);              // true, since 5 is not equal to 3
    bool public isGreaterThan = (10 > 5);          // true, since 10 is greater than 5
    bool public isLessThan = (3 < 8);               // true, since 3 is less than 8
    bool public isGreaterOrEqual = (6 >= 6);        // true, since 6 is equal to 6
    bool public isLessOrEqual = (9 <= 10);          // true, since 9 is less than 10

    // Logical operators
    bool public andOperator = true && false;      // false, since both must be true for '&&' to
    bool public orOperator = true || false;          // true, since one is true for '||' to
    bool public notOperator = !true;                 // false, since !true is false
}
```

The image displays two side-by-side screenshots of the Ethereum IDE interface on the Remix platform, showing different sets of smart contract functions.

Left Screenshot:

- Balance:** 0 ETH
- additionResult**: O: uint256: 8
- andOperator**: O: bool: false
- divisionResult**: O: uint256: 8
- isEqual**: O: bool: true
- isGreaterOrEq...**: O: bool: true
- isGreaterThan**: O: bool: true
- isLessOrEqual**: O: bool: true
- isLessThan**: O: bool: true
- isNotEqual**: O: bool: true
- moduloResult**

Right Screenshot:

- isGreaterOrEq...**: O: bool: true
- isGreaterThan**: O: bool: true
- isLessOrEqual**: O: bool: true
- isLessThan**: O: bool: true
- isNotEqual**: O: bool: true
- moduloResult**: O: uint256: 3
- multiplication...**: O: uint256: 12
- notOperator**: O: bool: false
- orOperator**: O: bool: true
- subtractionRe...**: O: uint256: 6

Common UI Elements:

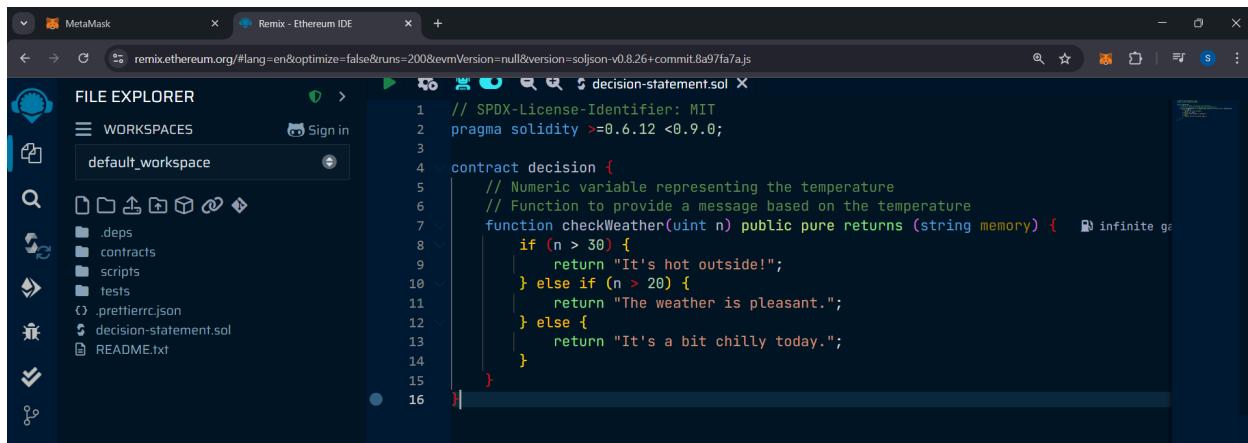
- Toolbar:** Includes icons for MetaMask, Remix logo, back/forward, and close.
- Address Bar:** Shows the URL: `remix.ethereum.org/#lang=en&optimize=false&runs=20`.
- Bottom Bar:** Contains "Scam Alert" and "Initialize as git repo" buttons.

C. Decision Making - if, if else, if else if

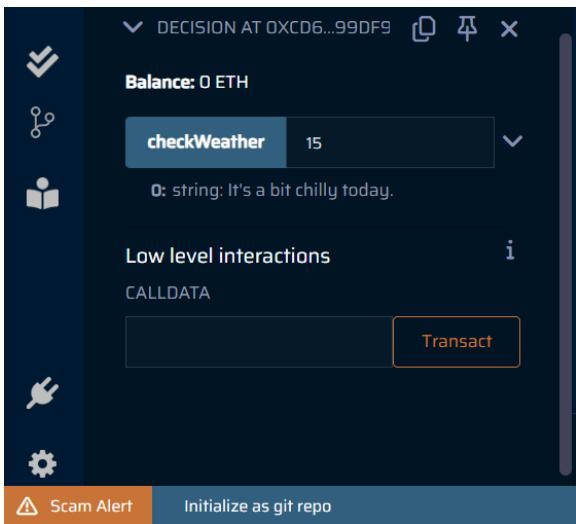
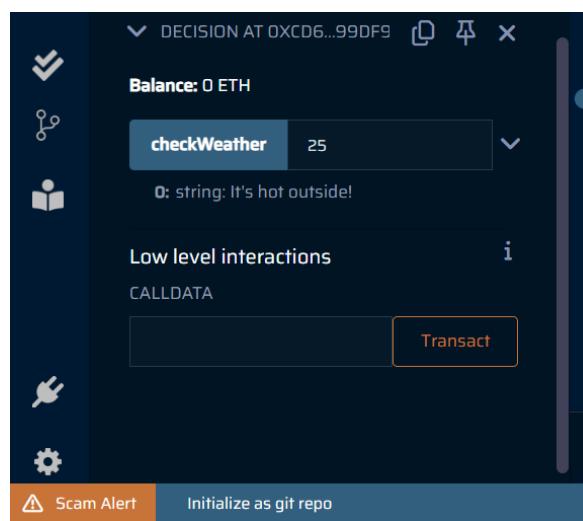
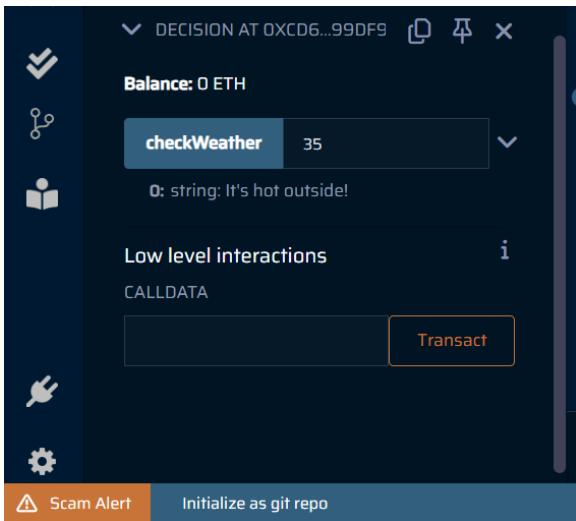
Solidity if, if-else, and if-else if statements enable condition-based decision-making. The if statement executes code when a condition is true, if-else adds an alternative for false conditions, and if-else if handles multiple sequential conditions. These constructs empower concise and adaptable logic in Ethereum smart contracts.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract decision {
    // Numeric variable representing the temperature
    // Function to provide a message based on the temperature
    function checkWeather(uint n) public pure returns (string memory) {
        if (n > 30) {
            return "It's hot outside!";
        } else if (n > 20) {
            return "The weather is pleasant.";
        } else {
            return "It's a bit chilly today.";
        }
    }
}
```



Output



D. Loops - for, while, and do while

In Solidity, loops streamline repetitive tasks. The 'for' loop efficiently iterates over a specific range. The 'while' loop repeats as long as a given condition holds true. The 'do-while' loop ensures a block of code runs at least once before checking the condition. These loops empower concise and effective iteration in Solidity smart contracts, enhancing efficiency and functionality.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;

contract Loop {

    // State variables to store the results of the sums
    uint public sum;
    uint public whileLoopSum;
    uint public doWhileLoopSum;

    // Function to calculate the sum of numbers from 1 to n using a for loop
    function calculateSum(uint n) public returns (uint) {
        // Initialize the sum variable
        sum = 0;

        // Loop to calculate the sum of numbers from 1 to n
        for (uint i = 1; i <= n; i++) {
            sum += i;
        }

        return sum;
    }

    // Function to calculate the sum of numbers from 1 to n using a while loop
    function calculateWhileLoopSum(uint n) public returns (uint) {
        // Initialize variables
        uint i = 1;
        whileLoopSum = 0;

        // Loop to calculate the sum using a while loop
        while (i <= n) {
            whileLoopSum += i;
            i++;
        }

        return whileLoopSum;
    }
}
```

```
// Function to calculate the sum of numbers from 1 to n using a do-while loop
function calculateDoWhileLoopSum(uint n) public returns (uint) {
    // Initialize variables
    uint i = 1;
    doWhileLoopSum = 0;

    // Loop to calculate the sum using a do-while loop
    do {
        doWhileLoopSum += i;
        i++;
    } while (i <= n);

    return doWhileLoopSum;
}
```

The screenshot shows the MetaMask wallet interface. On the left is a sidebar with various icons: a gear for settings, a plug for network selection, a checkmark for approvals, a magnifying glass for search, a double arrow for transfers, a lock for security, and a document for contracts. The main area is titled "DEPLOY & RUN TRANSACTIONS". It shows a "Deployed Contracts" section with one item: "LOOP AT 0XAEO...96B8B (M)". Below this, the "Balance" is listed as "0 ETH". There are three orange buttons for interacting with the deployed contract: "calculateDoW...", "calculateSum", and "calculateWhile...". Underneath these are three blue buttons for specific functions: "doWhileLoopS...", "sum", and "whileLoopSum". Each button has a value "0: uint256: 55" displayed next to it. At the bottom of the interface, there are buttons for "Scam Alert", "Initialize as git repo", and a green "Did you know..." button.

Practical 7: Write the solidity Code and demonstrate the following:

- Functions
 - View Functions
 - Pure Functions
 - Cryptographic Function
-

A. Functions (Non-View and Non-Pure)

- A regular function (without view or pure modifiers) can both read and modify the blockchain state.
- Regular functions are used when you need to update or change something on the blockchain (e.g., transferring tokens, updating balances, etc.).

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

contract Example {
    uint public balance = 1000;

    // Regular function: This modifies the state
    function deposit(uint amount) public {
        balance += amount; // This changes the state
    }
}
```

Output

The image shows two side-by-side screenshots of a blockchain interface, likely from a development environment like Truffle or Remix. Both screens display a contract with a single variable: `balance`. In the first screenshot, the `balance` is listed as `0: uint256: 1000`. Below it, there are two buttons: `deposit` (orange) and `balance` (blue). The second screenshot shows the same interface after a transaction. The `balance` value has changed to `6620`, and the `deposit` button now has a parameter input field containing `6620`. The `balance` button remains blue. Both screenshots include a sidebar with various icons and a footer with buttons for `Scam Alert`, `Initialize as git repo`, and `Did`.

B. View Functions

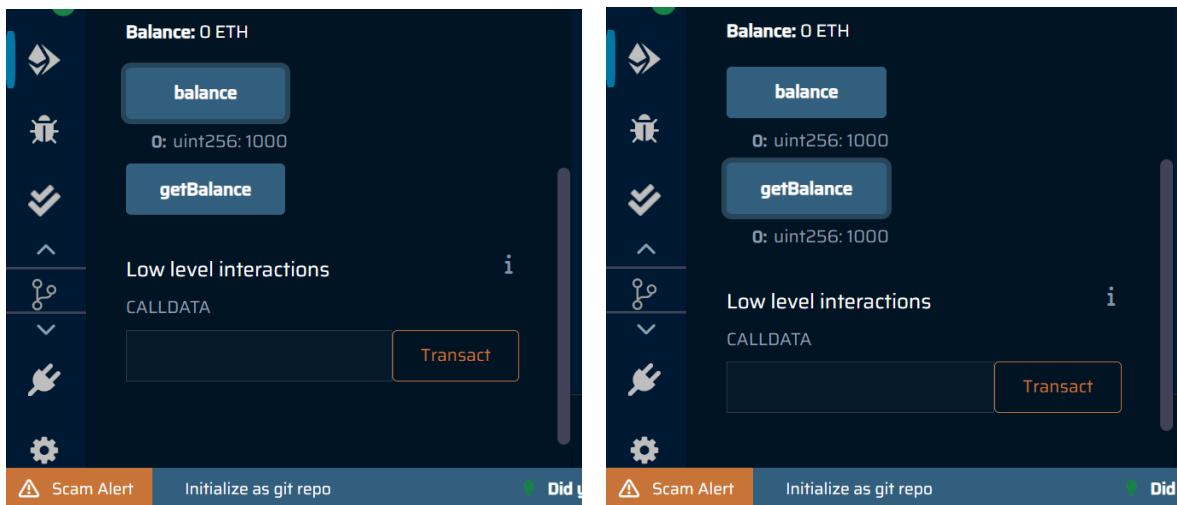
- A view function is one that does not modify the blockchain state. It can only read the state variables and return values, but it cannot change anything on the blockchain.
- A view function that returns the current value of a state variable without modifying it.

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

contract Example {
    uint public balance = 1000;

    // View function: This reads the state, but does not modify it
    function getBalance() public view returns (uint) {
        return balance; // This function only reads the state
    }
}
```

Output



C. Pure Functions

A pure function is a function that neither reads nor modifies the blockchain state. It operates solely on the provided arguments and returns a result based on them.

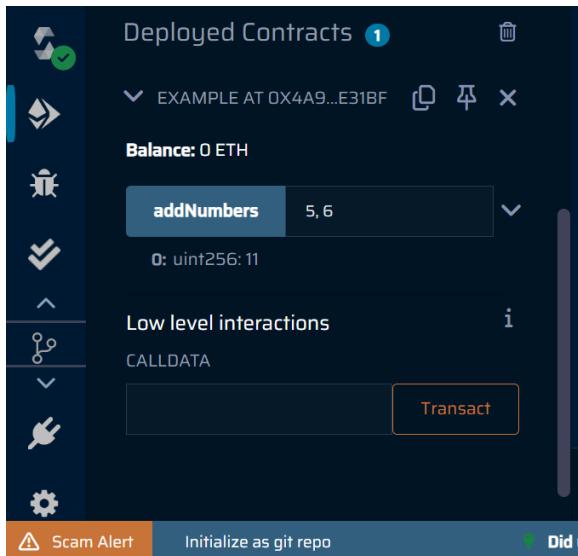
Use pure functions when you need to perform computations or transformations based solely on the function arguments, without accessing or modifying any state.

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

contract Example {

    // Pure function: This does not read or modify the state
    function addNumbers(uint a, uint b) public pure returns (uint) {
        return a + b; // This function just performs a calculation
    }
}
```

Output



D. Cryptographic Function

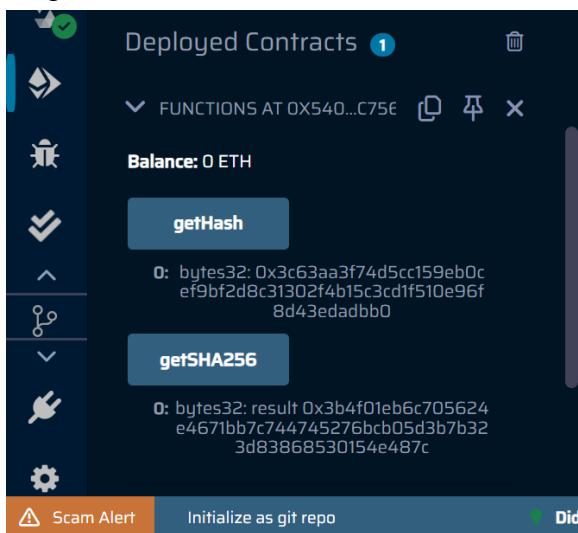
Cryptographic functions in Solidity typically involve operations such as hashing or verifying digital signatures. These are built-in functions that perform cryptographic operations and are often used for security purposes (e.g., hashing data, signing messages, etc.).

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

contract Functions {
    // Function to hash data using keccak256
    function getSHA256() public pure returns(bytes32 result){
        return sha256("6620");
    }

    // Function to hash data using keccak256
    function getHash() public pure returns (bytes32) {
        return keccak256("6620");
    }
}
```

Output



Practical 8: Write the solidity Code and demonstrate the following:

- a. Contracts
 - b. Inheritance
 - c. Constructors
 - d. Interfaces
-

A. Contracts

A Contract in Solidity is similar to a Class in C++. Contract has the following properties.

- Constructor – A special function declared with the 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.

B. Inheritance

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

contract C {
    uint private data; // Private state variable
    uint public info; // Public state variable

    // Constructor
    constructor() { 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 virtual 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() {
        c = new C();
    }

    function getComputedResult() public {
        result = compute(3, 5);
    }

    function getResult() public view returns (uint) {
        return result;
    }

    function getData() public view override returns (uint) {
        return c.getData();
    }
}
```

The image displays two identical-looking interfaces for managing deployed Ethereum contracts. Both interfaces have a dark blue header bar with a sidebar containing various icons. The main area shows a list of deployed contracts.

Left Panel:

- Deployed Contracts:** Shows 1 contract.
- Contract Address:** C AT 0xE5F...78E22 (MEMOFR)
- Balance:** 0 ETH
- Contract Methods:**
 - updateData**: uint256 a
 - getData**: uint256: 0
 - info**: uint256: 10
- Low level interactions**: CALldata
- Transact** button

Right Panel:

- Deployed Contracts:** Shows 1 contract.
- Contract Address:** C AT 0xE5F...78E22 (MEMOFR)
- Balance:** 0 ETH
- Contract Methods:**
 - updateData**: 6620
 - getData**: uint256: 6620
 - info**: uint256: 10
- Low level interactions**: CALldata
- Transact** button

Bottom Bar:

- Scam Alert** icon
- Initialize as git repo** button

C. Constructors

- Constructor is a special function declared using the 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.

D. Interfaces

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

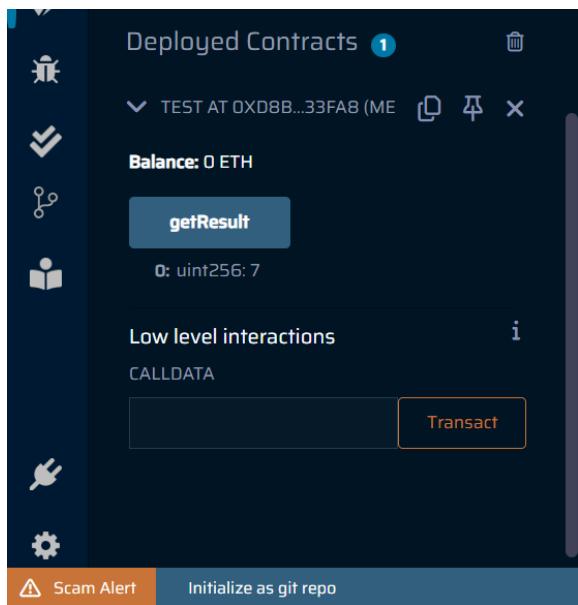
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface Calculator { // Define the Calculator interface
    function getResult() external pure returns (uint);
}

// Implement the Calculator interface in the Test contract
contract Test is Calculator {
    constructor() {}

    // Implementing the getResult function from the Calculator interface
    function getResult() external pure override returns (uint) {
        uint a = 5;
        uint b = 2;
        return a + b;
    }
}
```

Output



Practical 9: Program a Simple contract that can get, increment and decrement the count store in the contract.

The `Counter` contract allows you to get, increment, and decrement a stored count. The `getCount` function retrieves the count, while `increment` and `decrement` adjust it by 1. The contract emits a `CountUpdated` event for transparency.

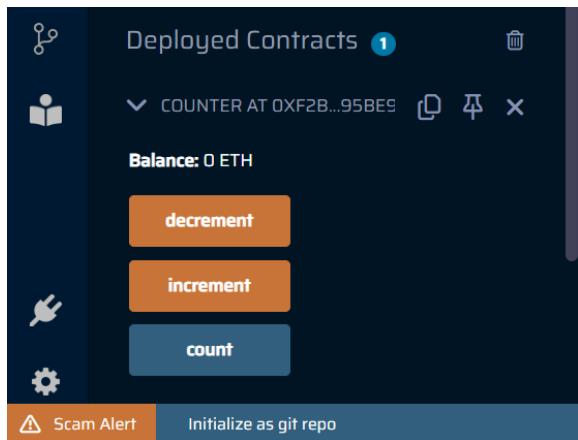
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Counter {
    uint public count;

    // Function to increment count by 1
    function increment() public {
        count += 1;
    }

    // Function to decrement count by 1
    function decrement() public {
        // Prevent decrementing if count is 0
        require(count > 0, "Count cannot be less than 0");
        count -= 1;
    }
}
```

Output



Getting count

Deployed Contracts 1

COUNTER AT 0xF2B...95BE9

Balance: 0 ETH

decrement

increment

count

0: uint256: 0

⚠ Scam Alert Initialize as git repo

Incrementing 4 times

Deployed Contracts 1

COUNTER AT 0xF2B...95BE9

Balance: 0 ETH

decrement

increment

count

0: uint256: 4

⚠ Scam Alert Initialize as git repo

Decrementing twice

Deployed Contracts 1

COUNTER AT 0xF2B...95BE9

Balance: 0 ETH

decrement

increment

count

0: uint256: 2

⚠ Scam Alert Initialize as git repo

What when count is 0 and we decrement

Deployed Contracts 1

COUNTER AT 0xF2B...95BE9

Balance: 0 ETH

decrement

increment

count

0: uint256: 0

⚠ Scam Alert Initialize as git repo

Deployed Contracts 1

COUNTER AT 0xF2B...95BE9

Balance: 0 ETH

decrement

increment

count

0: uint256: 0

✗ [vm] from: 0x5B3...eddC4 to: Counter.decrement() 0xf2B...95BE9 value: 0 wei
data: 0x2ba...eceb7 Logs: 0 hash: 0x436...9123d
transact to Counter.decrement errored: Error occurred: revert.

revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "Count cannot be Less than 0".
If the transaction failed for not having enough gas, try increasing the gas limit gently.

0 Listen on all transactions Filter with

Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.

⚠ Scam Alert Initialize as git repo

Practical 10: Write a simple smart contract to calculate the ‘number of ethers’ for the transaction of gas limit for the given scenario.

- a. Mint Your Own Coins and Make Simple Transactions With Solidity
 - b. Creating ERC -20 Token
-

A. Mint Your Own Coins and Make Simple Transactions With Solidity

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;
    event Sent(address from, address to, uint amount);

    constructor(){
        minter = msg.sender;
    }

    function mint(address receiver, uint amount)public{
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public{
        require(amount <= balances[msg.sender],"balances is not enough");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver,amount);
    }
}
```

The screenshot shows the Remix Ethereum IDE interface. On the left is the File Explorer sidebar with a list of contracts: .deps, contracts, artifacts, 1_Storage.sol, 2_Owner.sol, 3_Ballot.sol, minting.sol, my_token.sol, scripts, tests, .prettierc.json, and README.txt. The main central area displays the Solidity code for the minting.sol contract. The code defines a Coin contract with a constructor that initializes the minter to the sender and sets initial gas limits. It includes two functions: mint, which adds tokens to a receiver's balance if the sender is the minter and the amount is less than 1e60, and send, which transfers tokens from the sender's balance to a receiver's balance if the sender has enough tokens.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;
    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "balances is not enough");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Compile

This screenshot shows the Solidity Compiler interface within the Remix Ethereum IDE. The left sidebar features a 'SOLIDITY COMPILER' section with a dropdown set to '0.8.26+commit.8a97fa7a'. Below it are checkboxes for 'Include nightly builds', 'Auto compile', and 'Hide warnings'. A 'Advanced Configurations' dropdown is also present. In the center, the Solidity code for minting.sol is displayed, identical to the one in the previous screenshot. A tooltip 'Ctrl+S to compile contracts/minting.sol' is visible over the code editor. The bottom part of the interface includes a search bar, transaction filters, and system status indicators.

Deploy

The screenshot shows the Remix Ethereum IDE interface. On the left, the sidebar displays the 'DEPLOY & RUN TRANSACTIONS' section, which includes fields for 'ENVIRONMENT' (set to 'Injected Provider - MetaMask'), 'ACCOUNT' (set to 'Sepolia (11155111) network' with address '0xD6e...1949b (0.04741293912...)'), 'GAS LIMIT' (set to 'Estimated Gas'), 'VALUE' (set to '0 Wei'), and a 'CONTRACT' dropdown set to 'Coin - contracts/minting.sol'. Below these are buttons for 'Deploy' (highlighted in orange) and 'Deploy - transact (not payable)'. A checkbox for 'Publish to IPFS' is also present. The main area contains the Solidity code for 'minting.sol', which defines a 'Coin' contract with 'mint' and 'send' functions. At the bottom of the code editor, there are AI-related buttons: 'sol-gpt <your Solidity question here>' and 'Type the library name to see available commands.' The status bar at the bottom right shows 'RemixAI Copilot (enabled)', the date '23-01-2025', and the time '09:20'.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;
    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e80);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount < balances[msg.sender], "balances is not enough");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Select Environment as injector provider and connect meta mask wallet

Copy deployment address

The screenshot shows the Remix Ethereum interface with the following details:

- Deploy & Run Transactions**:
 - ENVIRONMENT: Injected Provider - MetaMask
 - ACCOUNT: Sepolia (11155111) network
 - GAS LIMIT: Estimated Gas (Custom: 3000000)
 - VALUE: 0 Wei
 - CONTRACT: Coin - contracts/minting.sol
 - Deploy button
 - Publish to IPFS checkbox
- Contract Source Code (minting.sol):**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;
    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e8);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "balances is not enough");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```
- Logs:**
 - web3.js
 - ethers.js
 - sol-gpt <your Solidity question here>

Type the library name to see available commands.
creation of Coin pending...
- Deployment Status:**
 - view on etherscan
 - [block:7551476 txIndex:34] from: 0xd6e...1949b to: Coin.(constructor) value: 0 wei data: 0x608...a0033 logs: 0x2f6...16ec0
 - Debug button
- System Bar:**
 - Scan Alert
 - Initialize as git repo
 - Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.
 - RemixAI Copilot (enabled)
- OS Taskbar:**
 - Type here to search
 - File, Mail, Microsoft Edge, Google Chrome, VS Code, GitHub
 - Construction on Sant... 09:22 23-01-2025 ENG

Deployment contract address

0x99829D95a6d949692Df31e797732ea7800a8efBE

Sepolia Testnet

Search by Address / Txn Hash / Block / Token

Overview State

[This is a Sepolia Testnet transaction only]

Transaction Hash: 0x3944b968bada2269665ba2eab8e04975e3cd60672ba4f5225f3e19544a72c3b7

Status: Success

Block: 7551476 | 1 Block Confirmation

Timestamp: 21 secs ago (Jan-23-2025 03:51:36 AM UTC)

Transaction Action: Call 0x6080604 Method by 0xD6e36e65...369D1949b

From: 0xD6e36e653a74298d805964F03E9A8Ab369D1949b

To: [0x99829d95a6d949692df31e797732ea7800a8efbe Created]

Value: 0 ETH

Transaction Fee: 0.00143971754430441 ETH

Gas Price: 3.527941249 Gwei (0.000000003527941249 ETH)

More Details: + Click to show more

Type here to search

Construction on Sant... 09:22 ENG 23-01-2025

6640_BCT - Google Docs | 6620_BT Journal - Google Docs | Remix - Ethereum IDE | MetaMask

DEPLOY & RUN TRANSACTIONS

Deployed Contracts

COIN 0X998...8efBE (BLOCKCHAIN)

Balance: 0 ETH

mint

receiver: 0xD6e36e653a74298d805964F03E9A8Ab369D1949b
amount: 10

send

receiver: address
amount: uint256

balances

minter

Low level interactions

CALldata

Estimated changes: No changes

Request from: remix.ethereum.org

Interacting with: 0x99829d95a6d949692df31e797732ea7800a8efbe

Network fee: 0.0002 SepoliaETH \$0.61

Speed: Market -15 sec

Cancel Confirm

Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.

Scan Alert Initialize as git repo

Type here to search

SENSEX +0.01% 09:30 ENG 23-01-2025

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar lists a deployed contract named 'COIN AT 0X998...8efBE (BLOCKCHAIN)' with a balance of 0 ETH. It includes sections for 'mint', 'send', and 'balances'. The 'mint' section has fields for 'receiver' (0xD6e36e653a74298d805964f03E9A8Ab369D1949b) and 'amount' (10), with buttons for 'Calldata', 'Parameters', and 'transact'. The 'send' section has fields for 'receiver' (address) and 'amount' (uint256), also with 'Calldata', 'Parameters', and 'transact' buttons. The 'balances' section has a 'minter' button. Below these are 'Low level interactions' and 'CALLDATA' buttons. The right side of the interface displays two code files: 'my_token.sol' and 'minting.sol'. The 'my_token.sol' file contains the following code:`1 require(amount <= balances[msg.sender], "Balances is not enough");
2 balances[msg.sender] -= amount;
3 balances[receiver] += amount;
4 emit Sent(msg.sender, receiver, amount);
5 }`Below the code, there are two entries in the transaction history:

- [block:7551476 txIndex:34] from: 0xd6e...1949b to: Coin.(constructor) value: 0 wei data: 0x608...a0833 logs: 0 hash: 0x2f6...16ec0
- [block:75514521 txIndex:7] from: 0xd6e...1949b to: Coin.mint(address,uint256) data: 0x40c...000a logs: 0 hash: 0x967...89319

A MetaMask window is open in the top right, showing 'Confirmed transaction' for Transaction 3. The system tray at the bottom right shows the date as 23-01-2025.

This screenshot is similar to the first one but includes a 'Transaction request' dialog box from MetaMask. The dialog shows the following details:

- Estimated changes: No changes
- Request from: remix.ethereum.org
- Interacting with: 0x99829...8efBE
- Network fee: 0.0002 SepoliaETH \$0.66
- Speed: Market -15 sec

At the bottom of the dialog are 'Cancel' and 'Confirm' buttons. The transaction history at the bottom of the Remix interface shows an error message for the 'send' transaction:

```
transact to coin.send errored: Error encoding arguments: Error: invalid BigNumber string (argument="value", value="3.99", code=INVALID_ARGUMENT)
call to Coin.balances

call [call] from: 0x06e36e653a74298d805964f03E9A8Ab369D1949b to: Coin.balances(address) data: 0x27e...1949b
transact to coin.send pending ...
```

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar lists a deployed contract named 'COIN AT 0X998...8EFB8E'. The 'mint' function has been called twice: once with receiver 0xD6e36e653a74298d805964F03E9A8Ab369D1949b and amount 10, and once with receiver 0x44611aDx075df45334dd97DD5E8A0E565e3Ad56 and amount 5. The 'send' function has also been called once with receiver 0x44611aDx075df45334dd97DD5E8A0E565e3Ad56 and amount 5. The 'balances' function has been called once with address 0xD6e36e653a74298d805964F03E9A8Ab369D1949b. The bottom right corner shows a confirmation message: 'Confirmed transaction Transaction 4 confirmed! View on Sepolia Etherscan'.

This screenshot is nearly identical to the one above, showing the same deployed contract 'COIN AT 0X998...8EFB8E' and its interactions. The 'mint' function was called twice with amounts 10 and 5 respectively, and the 'send' function was called once with amount 5. The 'balances' function was called once. The bottom right corner again shows a confirmation message: 'Confirmed transaction Transaction 4 confirmed! View on Sepolia Etherscan'.

COIN AT 0X998...8EFBE (BLOCKCHAIN)

Balance: 0 ETH

mint

receiver: 0xD6e36e653a74298d805964F03E9A8Ab369D1949b

amount: 10

send

receiver: 0x44611aDa075df45334dd97DD5E6A0E565eE3Ad56

amount: 5

balances

: 0xD6e36e653a74298d805964F03E9A8Ab369D1949b

0: uint256: 5

minter

This screenshot shows a user interface for interacting with a blockchain contract named 'COIN'. The interface is dark-themed with orange and blue buttons. At the top, it displays the contract address '0X998...8EFBE (BLOCKCHAIN)' and a balance of '0 ETH'. Below this, there are three main sections: 'mint', 'send', and 'balances'. The 'mint' section allows the user to send 10 tokens to a specified receiver (0xD6e36e653a74298d805964F03E9A8Ab369D1949b). The 'send' section allows the user to send 5 tokens to another receiver (0x44611aDa075df45334dd97DD5E6A0E565eE3Ad56). The 'balances' section shows the current balance of the contract (5 tokens) and provides a 'call' button to interact with the contract's logic. A large blue 'minter' button is located at the bottom left.

Screenshot of the MetaMask extension interface in a browser window.

The top navigation bar shows tabs for "6640_BCT - Google Docs", "6620_BT Journal - Google Docs", "Remix - Ethereum IDE", and "MetaMask".

The MetaMask interface displays a "Portfolio" balance of **+\$0.00 (+0.00%)**. Below the portfolio are buttons for "Buy & Sell", "Swap", "Bridge", "Send", and "Receive".

A modal window titled "Ready to bridge?" with the sub-instruction "Move across 9 chains, all within your wallet" is visible.

The main content area has tabs for "Tokens" and "NFTs", with "Activity" selected. The activity log shows the following transactions:

- Jan 23, 2025:
 - Send** Confirmed: -0 SepoliaETH, -0 SepoliaETH
 - Mint** Confirmed: -0 SepoliaETH, -0 SepoliaETH
 - Contract deployment** Confirmed: -0 SepoliaETH, -0 SepoliaETH
 - Send ZUN** Confirmed: -0 SepoliaETH
 - Contract deployment** Confirmed: -0 SepoliaETH

A reminder message at the bottom right says "Back up your Secret Recovery Phrase to keep your wallet and funds secure." with a "Back up now" button.

The taskbar at the bottom includes icons for File Explorer, Mail, Task View, Edge, Google Chrome, and File Explorer again. A "MetaMask support" link is also present.

The system tray shows battery status (Air: Poor), network signal, language (ENG), date (23-01-2025), and time (09:36).

B. Creating ERC -20 Token

Open MetaMask Wallet

Get ETH from Google Cloud seplia Network test network faucet

<https://cloud.google.com/application/web3/ethereum/sepolia>

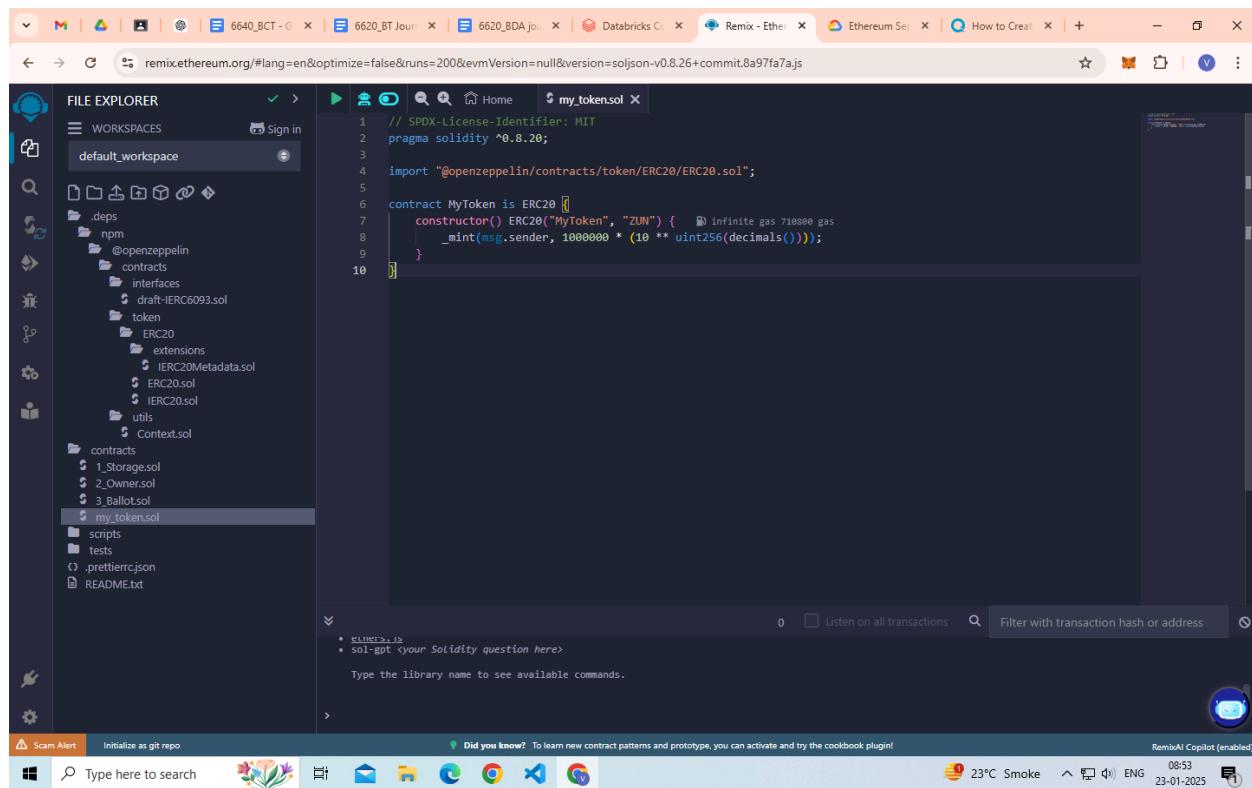
REF:

<https://www.quicknode.com/guides/ethereum-development/smart-contracts/how-to-create-and-deploy-an-erc20-token>

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "ZUN") {
        _mint(msg.sender, 1000000 * (10 ** uint256(decimals())));
    }
}
```



The screenshot shows the Remix IDE interface. On the left, the Solidity Compiler sidebar is open, showing the version 0.8.26+commit.8a97fa7a. It includes options for 'Include nightly builds', 'Auto compile', and 'Hide warnings'. A button labeled 'Compile my_token.sol' is highlighted with a tooltip 'Ctrl+S to compile contracts/my_token.sol'. Below this is a 'Compile and Run script' button. The main workspace displays a Solidity code editor with the file 'my_token.sol' containing the following code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "ZUN") {
        _mint(msg.sender, 100000 * (10 ** uint256(decimals())));
    }
}
```

At the bottom of the screen, a Windows taskbar is visible with various icons and a system tray showing the date and time.

The screenshot shows the Remix IDE interface with the 'DEPLOY & RUN TRANSACTIONS' sidebar open. The environment is set to 'Remix VM (Cancun)'. The account selected is '0x5B3...eddC4 (100 ether)'. The gas limit is set to 'Estimated Gas' with a value of 3000000. The contract being deployed is 'MyToken - contracts/my_token.sol' with an 'evm version cancan'. The 'Deploy' button is highlighted. The right side of the screen shows the same Solidity code editor and the bottom taskbar.

A separate window titled 'MetaMask' is overlaid on the screen, prompting the user to 'Connect with MetaMask'. It asks for permission to 'See your accounts and suggest transactions' and 'Use your enabled networks'. Buttons for 'Cancel' and 'Connect' are present.

The screenshot shows the Remix Ethereum IDE interface. On the left, the sidebar includes sections for Deploy & Run Transactions, Environment (Sepolia network), Account (0xD6e...1949b), Gas Limit (Custom, 300000), Value (0 Wei), and CONTRACT (MyToken - contracts/my_token.sol). The main area displays the Solidity code for the MyToken contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "ZUN") {
        _mint(msg.sender, 100000 * (10 ** uint256(decimals())));
    }
}
```

Below the code, there are two deployment buttons: Deploy and Deploy - transact (not payable). The Deploy button is highlighted. The status bar at the bottom indicates "Scam Alert" and "Initialize as git repo".

The screenshot shows the Remix Ethereum IDE interface with the MetaMask extension open. The MetaMask window title is "Deploy a contract". It displays the following information:

- Estimated changes: No changes
- Request from: remix.ethereum.org
- Network fee: 0.0026 SepoliaETH \$0.49
- Speed: Market ~15 sec

At the bottom of the dialog are "Cancel" and "Confirm" buttons. The status bar at the bottom indicates "Scam Alert" and "Initialize as git repo".

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is open, showing the environment set to 'Injected Provider - MetaMask' on the 'Sepolia (11155111) network'. The account selected is '0xD6e...1949b (0.0475607111...)'. The gas limit is set to 'Estimated Gas'. The value is set to 0 Wei. The contract being deployed is 'MyToken - contracts/my_token.sol'. The EVM version is set to 'can-can'. A 'Deploy' button is visible. Below the sidebar, there's a section for 'Transactions recorded' with options to 'Run transactions using the latest compilation result', 'Save', and 'Run'. At the bottom of the sidebar, there are buttons for 'At Address' and 'Load contract from Address'. The main central area displays the Solidity code for the 'MyToken' contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

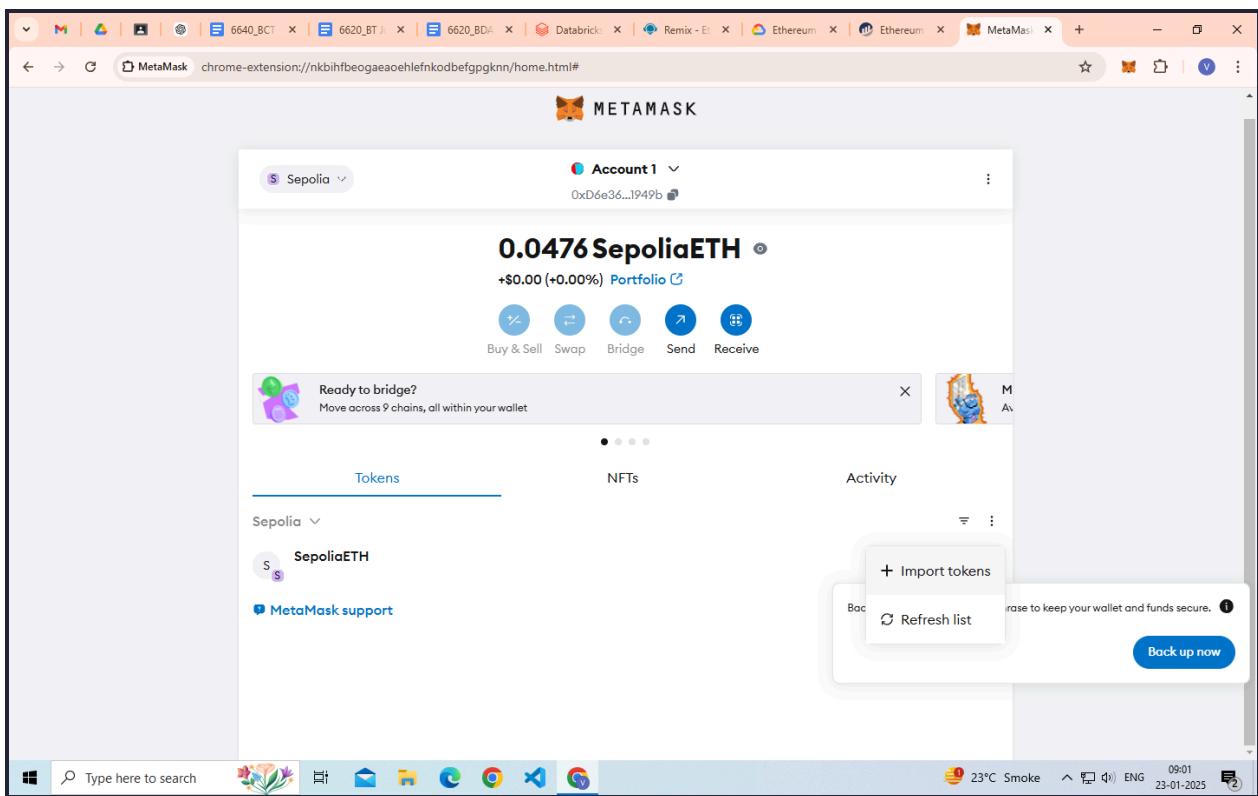
contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "ZUN") {
        _mint(msg.sender, 100000 * (10 ** uint256(decimals())));
    }
}
```

Below the code, a transaction confirmation message is shown in a separate window:

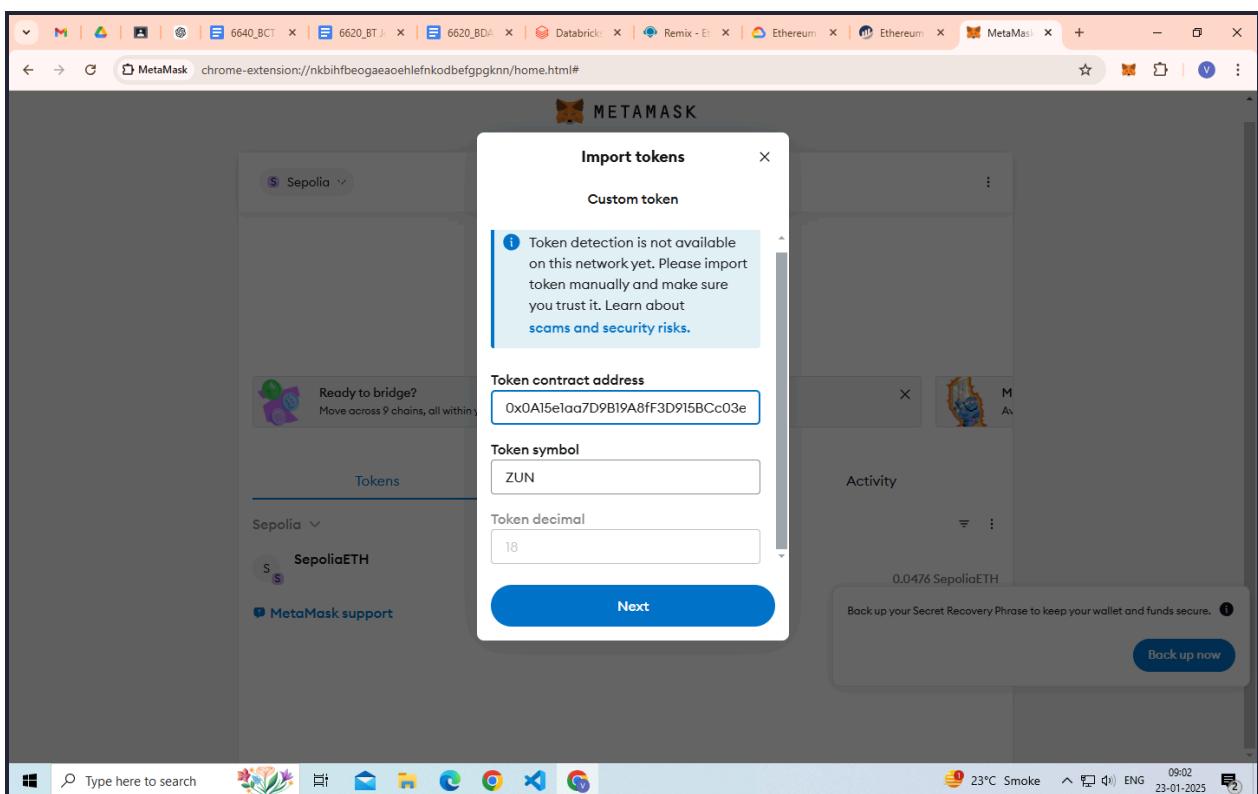
Confirmed transaction
Transaction 0 confirmed! View on Sepolia Etherscan

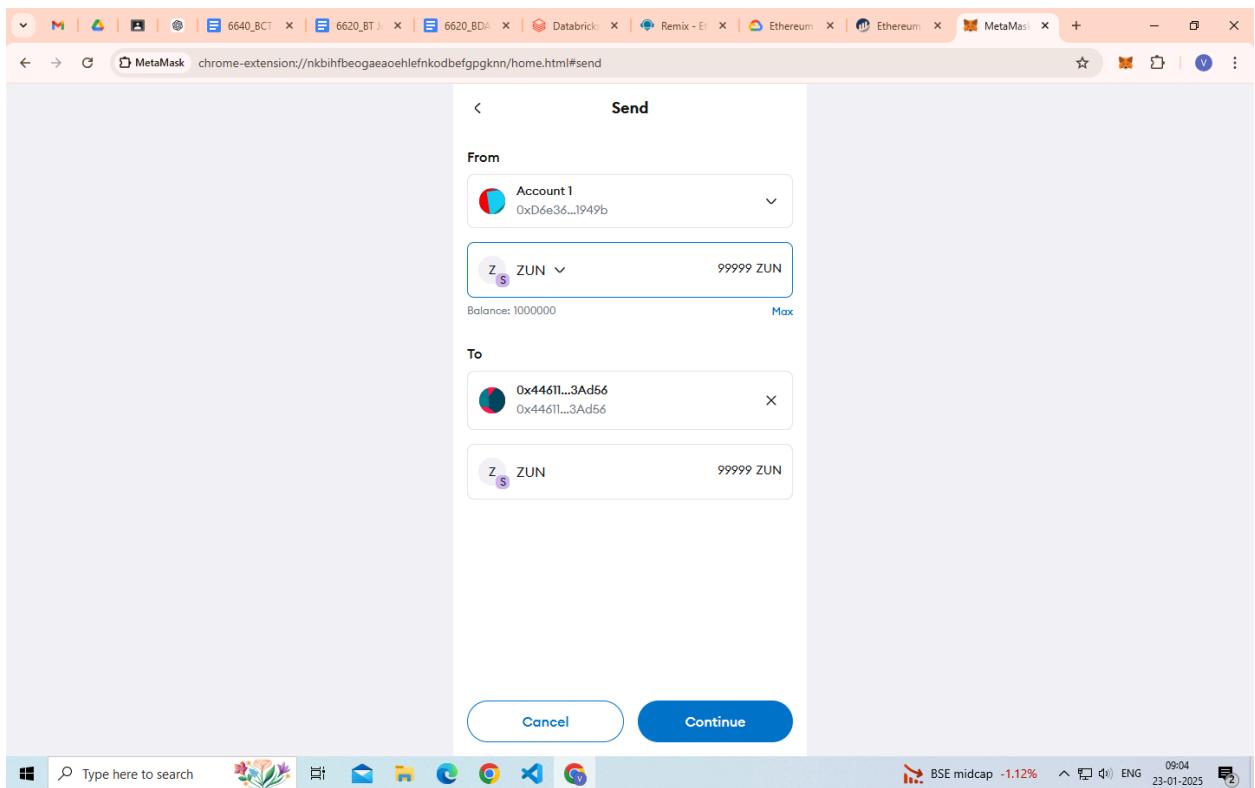
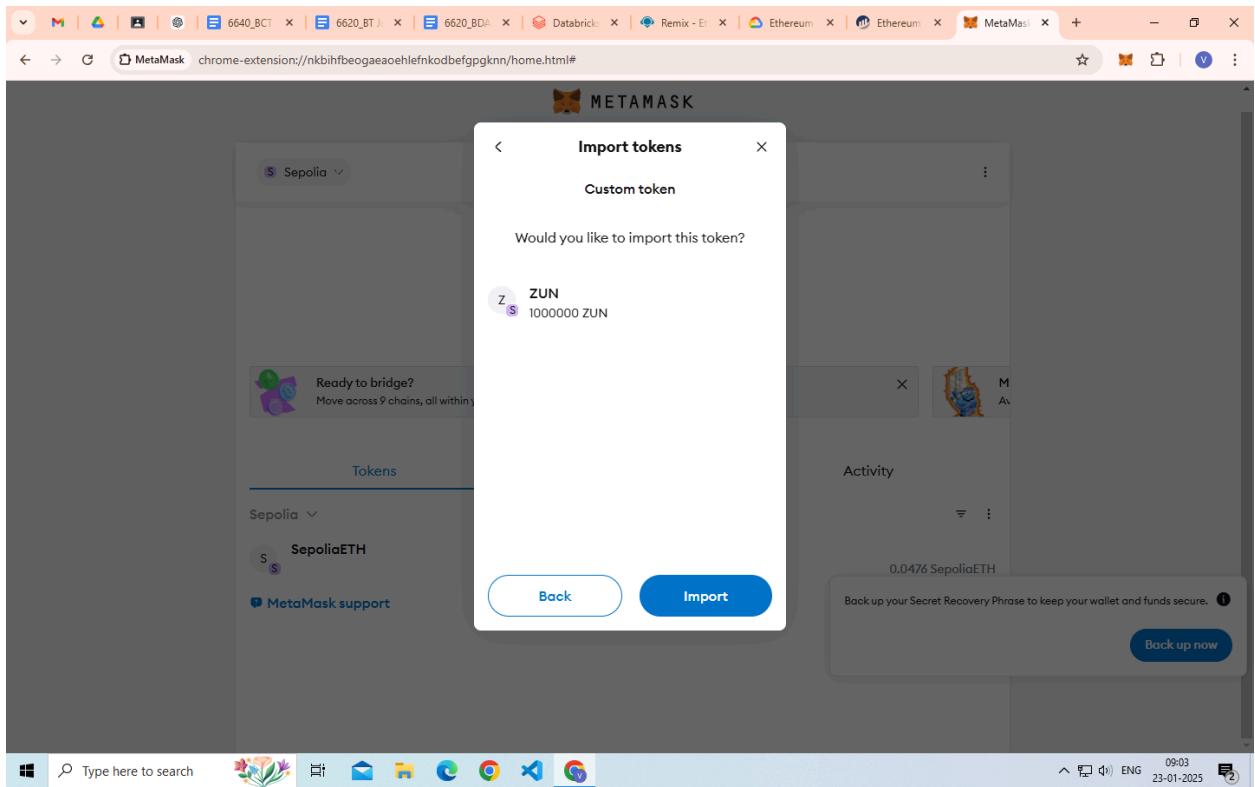
This screenshot is nearly identical to the one above, showing the same interface and Solidity code for the 'MyToken' contract. However, the 'Deployed Contracts' section at the bottom of the sidebar now lists a single contract: 'MYTOKEN AT 0x0A1...088A6'. The status of this contract is 'Deployed'.

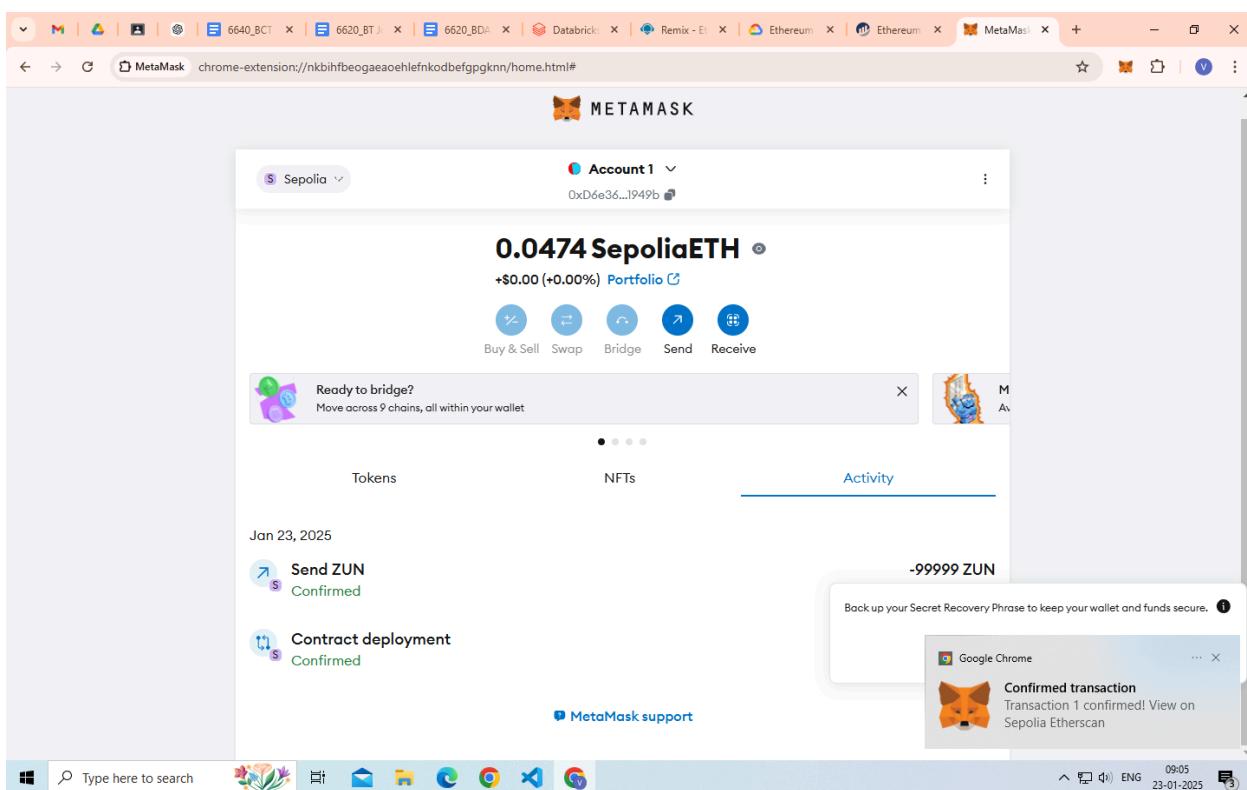
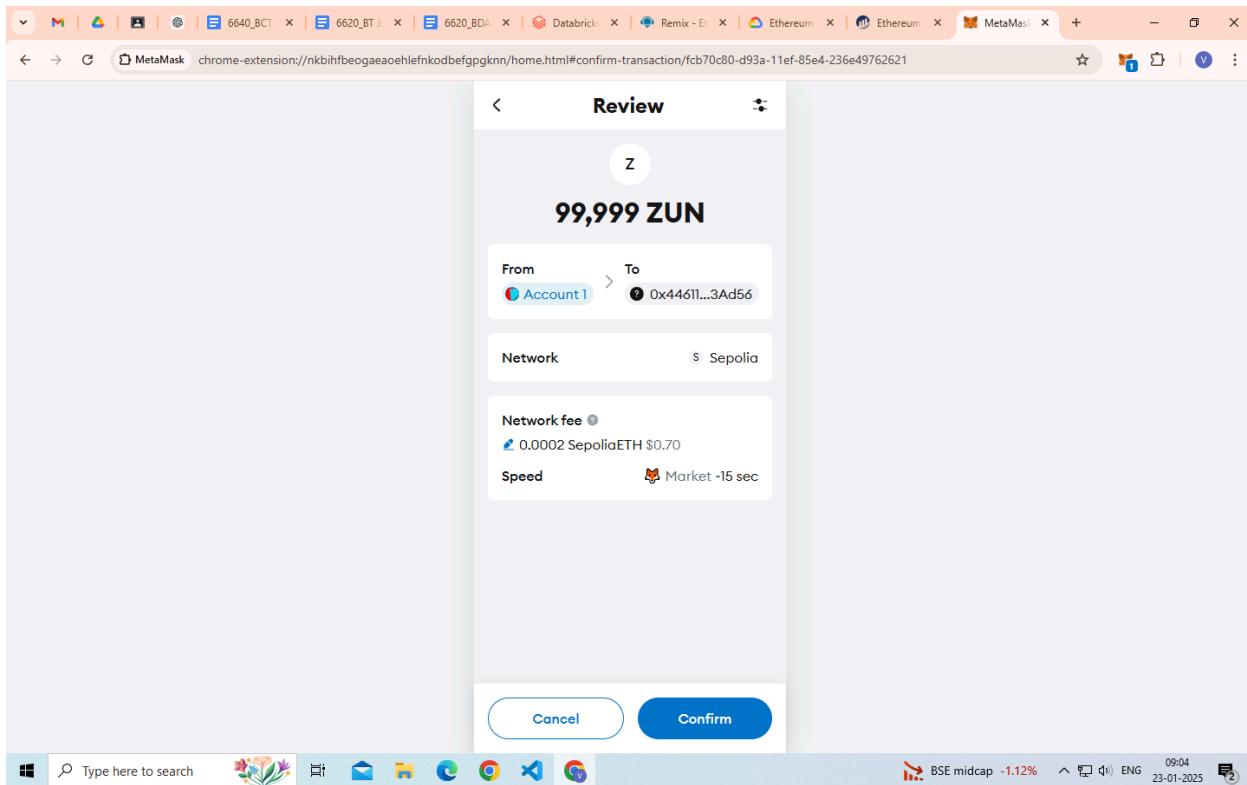
0x0A15e1aa7D9B19A8fF3D915BCc03e4F07060bBA6

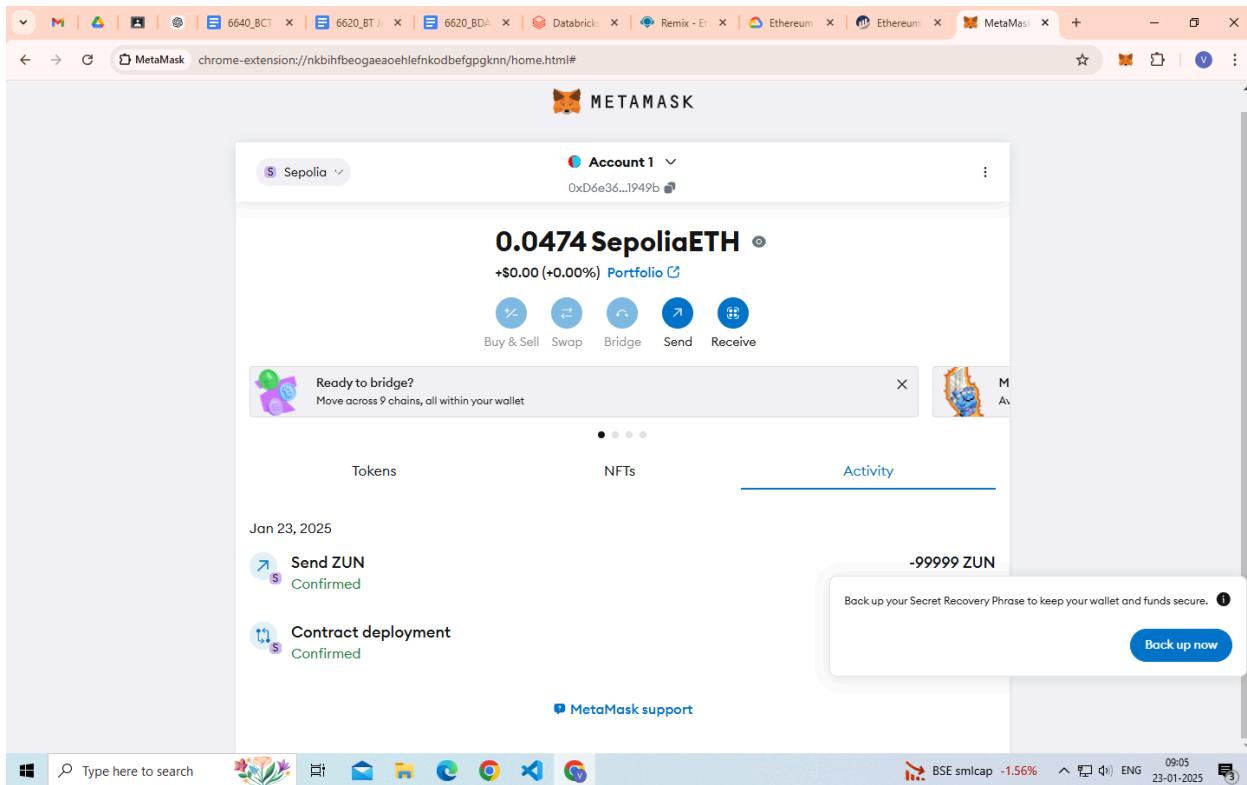


Enter Remix transaction ID

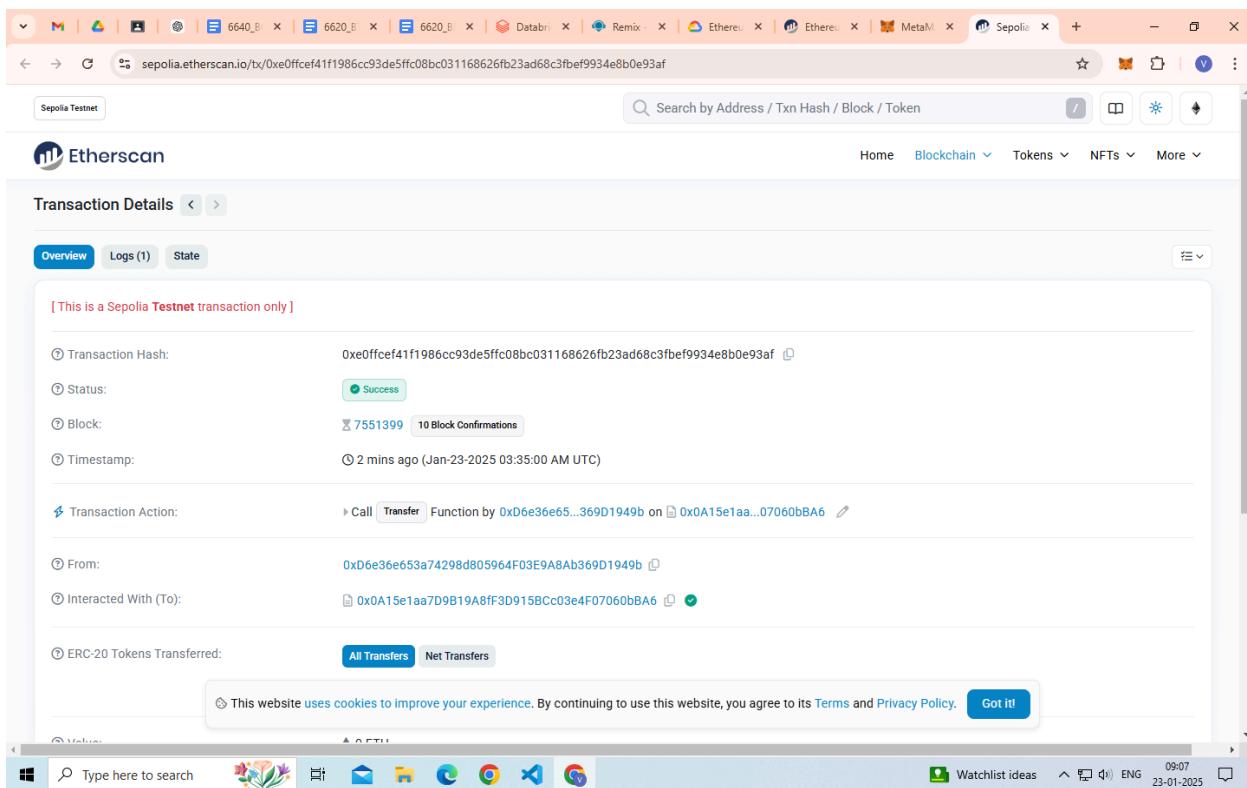








Click on Meta mask confirm transaction notification



Sepolia Testnet | sepolia.etherscan.io/bc/0xe0ffcef41f1986cc93de5ffc08bc031168626fb23ad68c3fbef9934e8b0e93af

Etherscan

Transaction Details

Overview Logs (1) State

[This is a Sepolia Testnet transaction only]

Transaction Hash: 0xe0ffcef41f1986cc93de5ffc08bc031168626fb23ad68c3fbef9934e8b0e93af ⓘ

Status: Success

Block: 7551399 10 Block Confirmations

Timestamp: 2 mins ago (Jan-23-2025 03:35:00 AM UTC)

Transaction Action: Call Transfer Function by 0xD6e36e65...369D1949b on 0xA15e1aa...07060bBA6 ⓘ

From: 0xD6e36e653a74298d805964f03e9a8ab369d1949b ⓘ

Interacted With (To): 0xA15e1aa7D9B19A8fF3D915BCC03e4F07060bBA6 ⓘ ✓

ERC-20 Tokens Transferred: All Transfers Net Transfers

This website uses cookies to improve your experience. By continuing to use this website, you agree to its Terms and Privacy Policy. Got It!

https://sepolia.etherscan.io/address/0xd6e36e653a74298d805964f03e9a8ab369d1949b

Sepolia Testnet | sepolia.etherscan.io/address/0xd6e36e653a74298d805964f03e9a8ab369d1949b#tokentxns

Address 0xD6e36e653a74298d805964f03e9a8ab369d1949b

Overview

ETH BALANCE ⚡ 0.047412939122668251 ETH

TOKEN HOLDINGS \$0.00 (2 Tokens)

More Info

TRANSACTIONS SENT
Latest: 9 mins ago ⓘ First: 18 mins ago ⓘ

FUNDED BY 0x42645cE4...Ea670f9b2 ⓘ at tx 0x25c6658679...

Multichain Info
N/A

Transactions Token Transfers (ERC-20)

Latest 3 ERC-20 Token Transfer Events

Transaction Hash	Method	Block	Age	From	To	Amount	Token
0xe0ffcef41f19...	Transfer	7551399	9 mins ago	0xD6e36e65...369D1949b ⓘ	OUT 0x44611aDa...65eE3Ad56 ⓘ	99,999 ⚡ \$0.00	ERC-20: MyToken
0xb174b31f456...	Transfer	7551369	15 mins ago	0x44611aDa...65eE3Ad56 ⓘ	IN 0xD6e36e65...369D1949b ⓘ	10,000	ERC-20: MyToken
0x87dbb9ec21...							ERC-20: MyToken

This website uses cookies to improve your experience. By continuing to use this website, you agree to its Terms and Privacy Policy. Got It!

Type here to search ⓘ Watchlist ideas ⓘ ENG 09:08 23-01-2025