RAMNIRANJAN JHUNJHUNWALA COLLEGE GHATKOPAR (W), MUMBAI - 400 086

DEPARTMENT OF INFORMATION TECHNOLOGY

2023 -2024

M.SC.(I.T.) PART I SEM II

RJSPITE203P

THEORY AND APPLICATIONS OF BLOCKCHAIN ASSIGNMENT

NAME. SANDESHKUMAR SINGH

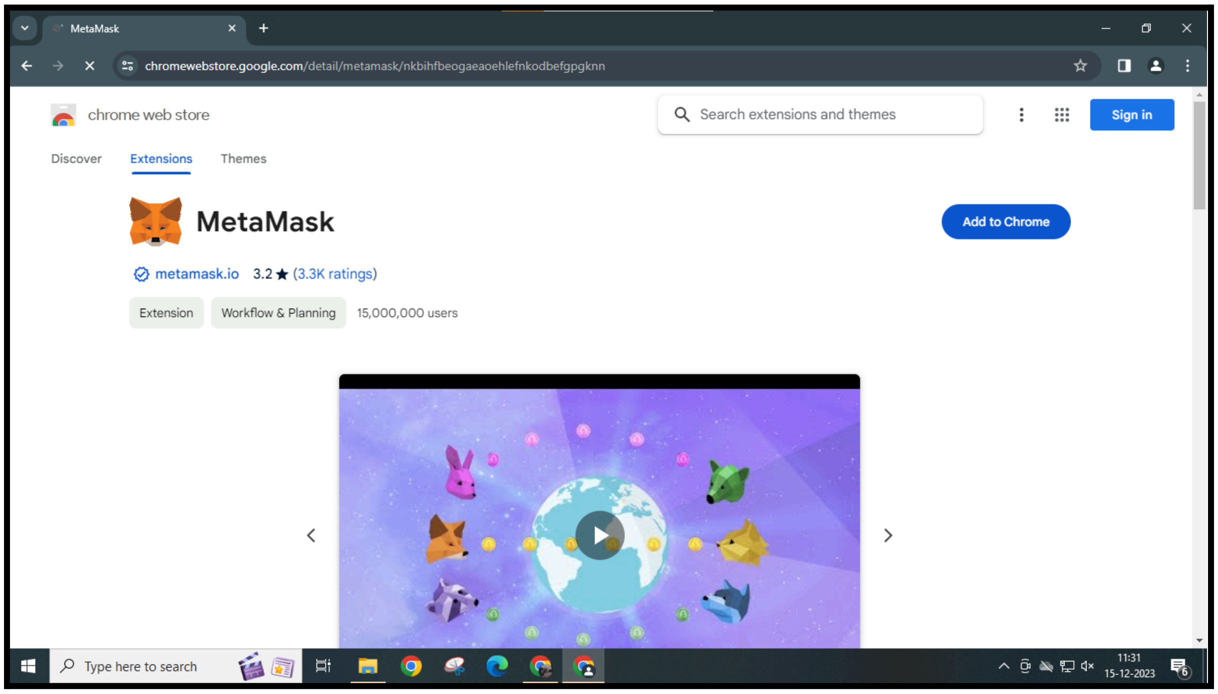
ROLL NO. 615

Index

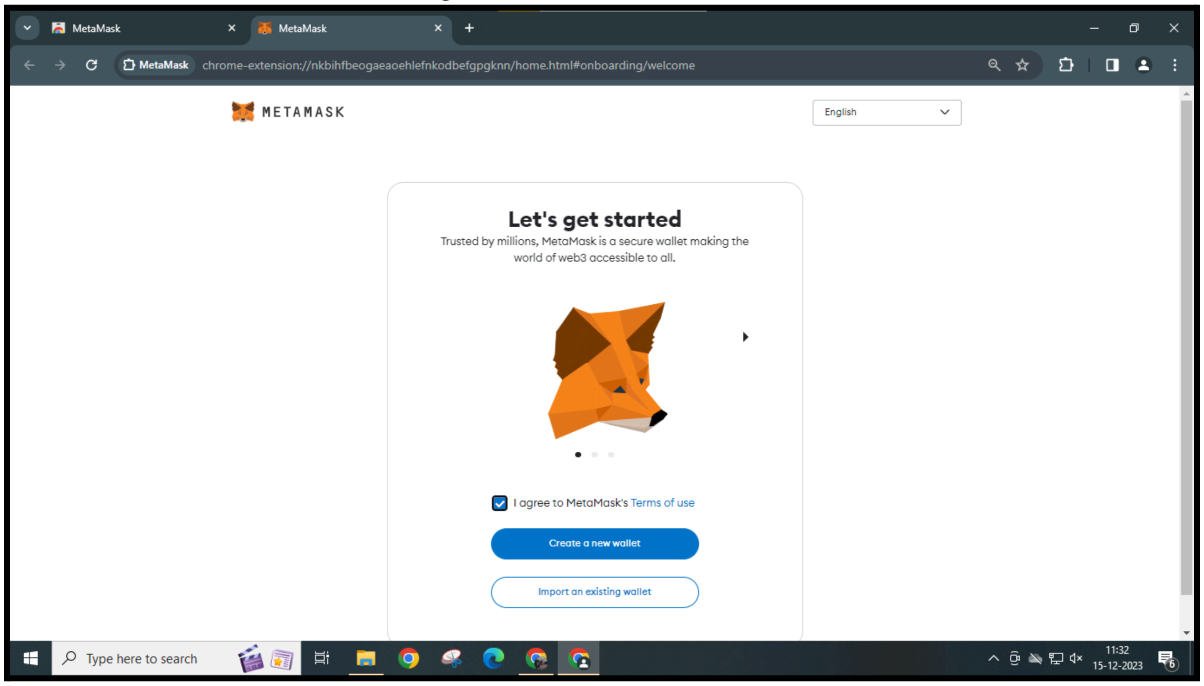
1. Demonstrate Working with MetaMask

a. Install Metamask ,Create account ,Deposit Ether

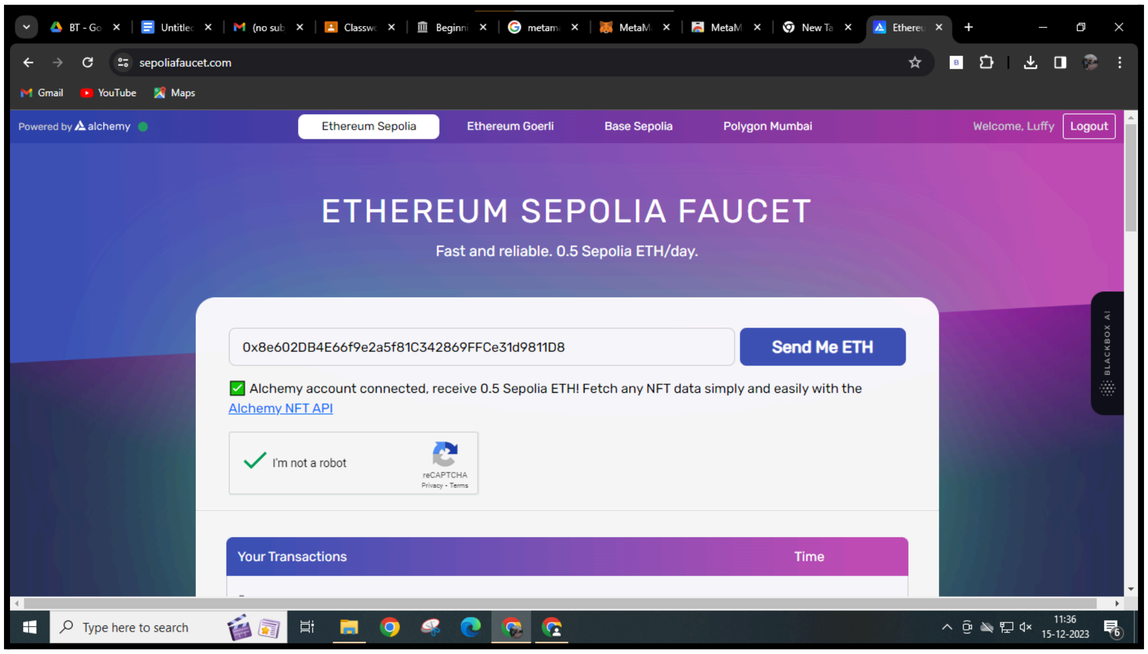
To Install search Metamask extension and add to chrome.



Create a wallet after adding an extension to chrome.


MetaMask
MetaMask
chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/home.html#onboarding/welcome
METAMASK
English
Let's get started
Trusted by millions, MetaMask is a secure wallet making the world of web3 accessible to all.
I agree to MetaMask's Terms of use
Create a new wallet
Import an existing wallet
Type here to search
11:32
15-12-2023

Now use Test Network Sepholia and add the eth into it.
To do that copy the public address and go to https://sepoliafaucet.com/ and signup and enter the address and click on send me ETH.



ETHEREUM SEPOLIA FAUCET

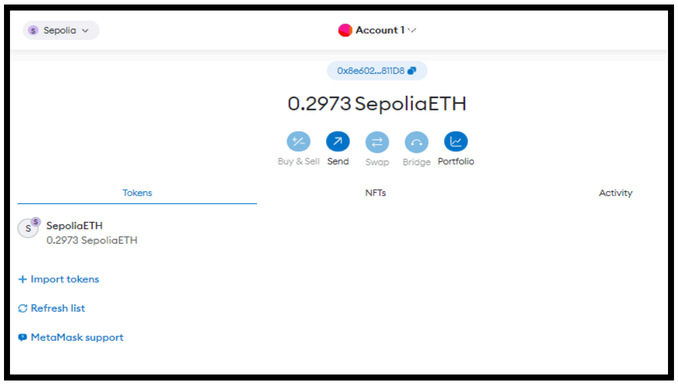
Fast and reliable. 0.5 Sepolia ETH/day.

0x8e602DB4E66f9e2a5f81C342869FFCe31d9811D8 — Send Me ETH

✓ Alchemy account connected, receive 0.5 Sepolia ETH! Fetch any NFT data simply and easily with the Alchemy NFT API

I'm not a robot

Your Transactions	Time

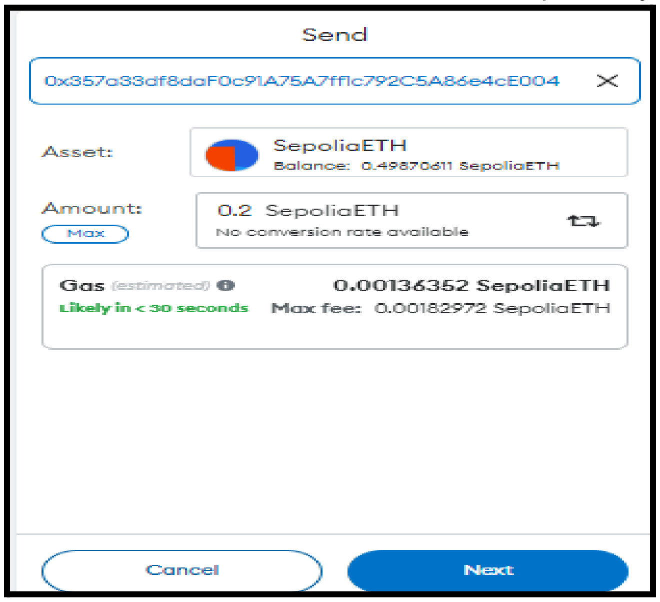


0.2973 SepoliaETH

SepoliaETH
0.2973 SepoliaETH

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



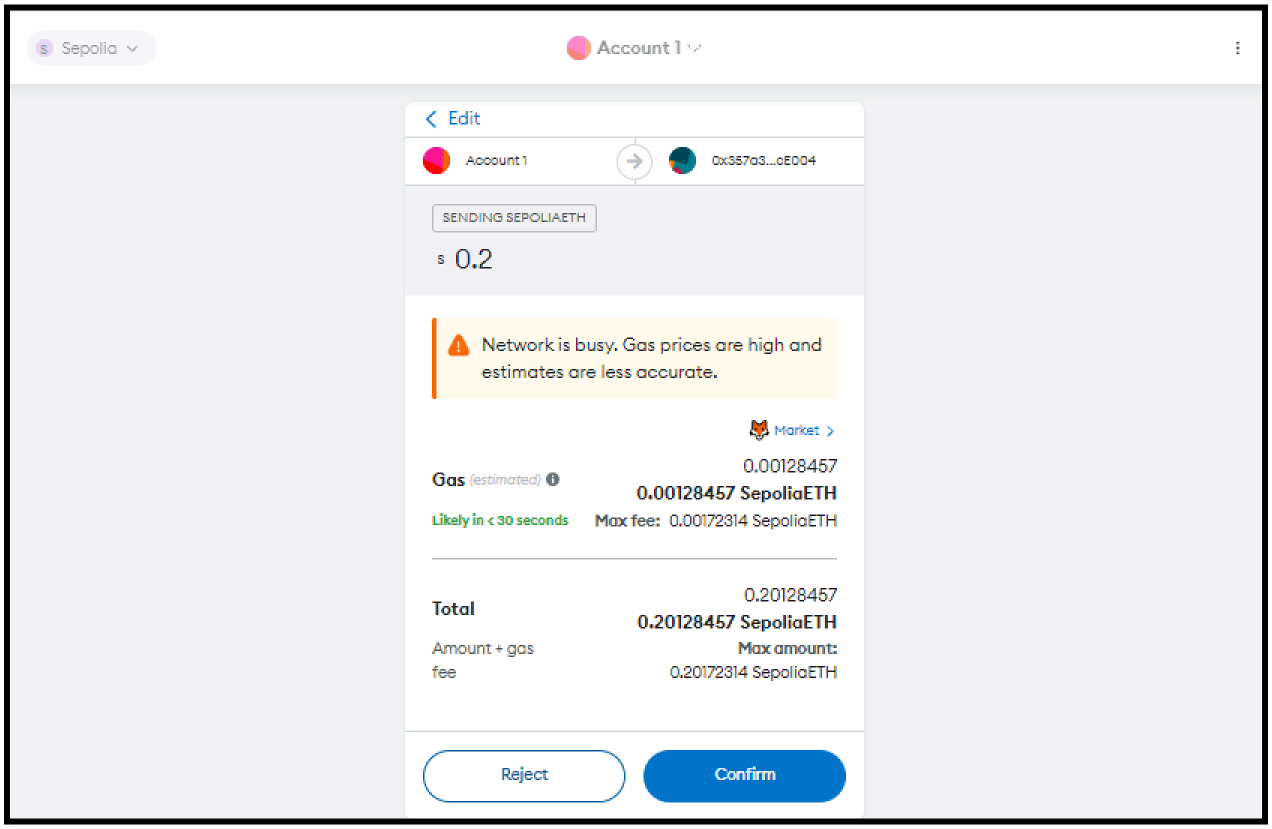
0x357a33df8daF0c91A75A7ff1c792C5A86e4cE004

Asset: SepoliaETH
Balance: 0.49870611 SepoliaETH

Amount: 0.2 SepoliaETH
No conversion rate available

Gas (estimated) 0.00136352 SepoliaETH
Likely in < 30 seconds Max fee: 0.00182972 SepoliaETH

Click on Next



Sepolia
Account 1

Edit
Account 1 → 0x357a3...cE004

SENDING SEPOLIAETH
0.2

Network is busy. Gas prices are high and estimates are less accurate.

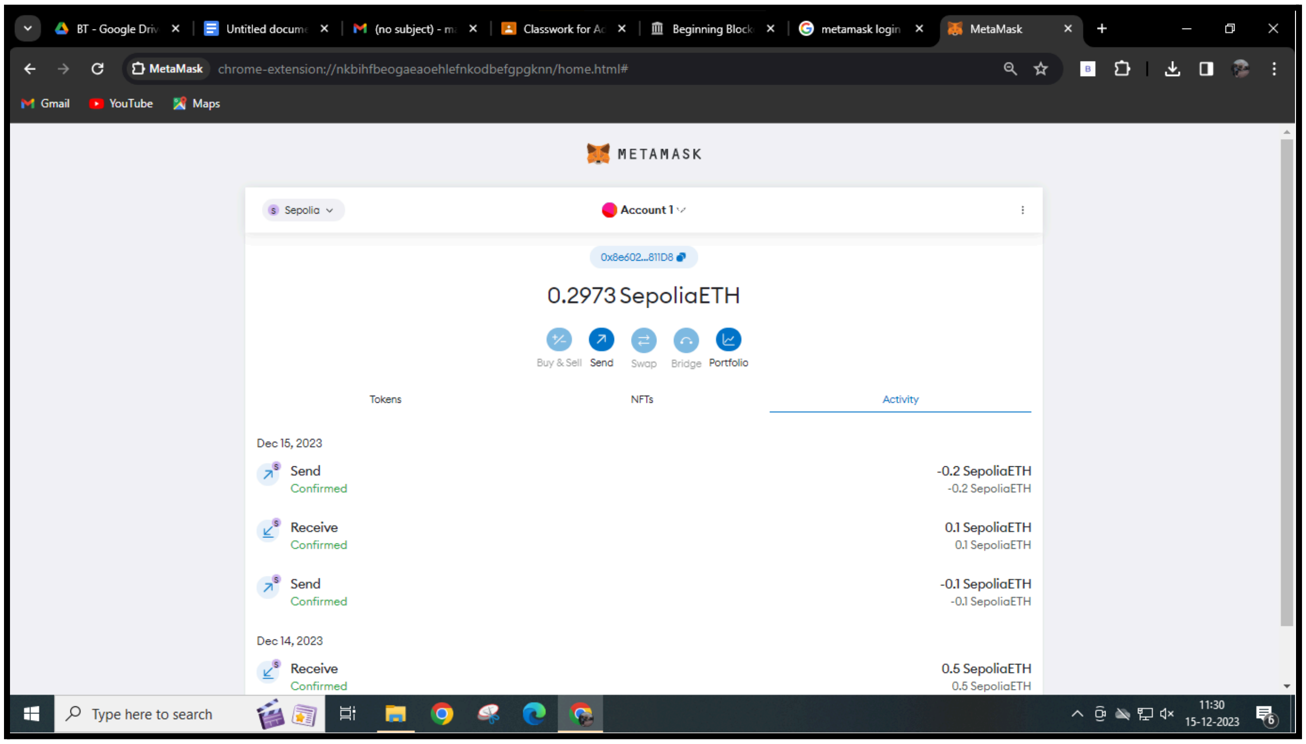
Market

Gas (estimated) 0.00128457
0.00128457 SepoliaETH
Likely in < 30 seconds Max fee: 0.00172314 SepoliaETH

Total 0.20128457
0.20128457 SepoliaETH
Amount + gas fee Max amount: 0.20172314 SepoliaETH

Reject Confirm

Click on Confirm and wait.



chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/home.html#

METAMASK

Sepolia
Account 1
0x8e602...811D8

0.2973 SepoliaETH

Buy & Sell | Send | Swap | Bridge | Portfolio

Tokens | NFTs | Activity

Dec 15, 2023

Send
Confirmed
-0.2 SepoliaETH
-0.2 SepoliaETH

Receive
Confirmed
0.1 SepoliaETH
0.1 SepoliaETH

Send
Confirmed
-0.1 SepoliaETH
-0.1 SepoliaETH

Dec 14, 2023

Receive
Confirmed
0.5 SepoliaETH
0.5 SepoliaETH

11:30
15-12-2023

2. Implementing cryptography Algorithm in Python:

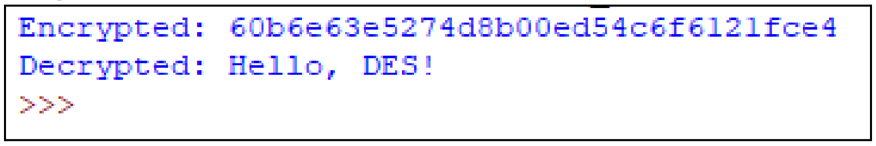
a. DES

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.

Source Code:

```
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)
    padded_message = pad(message.encode('utf-8'), 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)
    return unpadded_message.decode('utf-8')

if __name__ == "__main__":
    key = generate_key()
    message = "Hello, DES!"

    ciphertext = encrypt(message, key)
    print("Encrypted:", ciphertext.hex())

    decrypted_message = decrypt(ciphertext, key)
    print("Decrypted:", decrypted_message)
```

Output:



```
Encrypted: 60b6e63e5274d8b00ed54c6f6121fce4
Decrypted: Hello, DES!
>>>
```

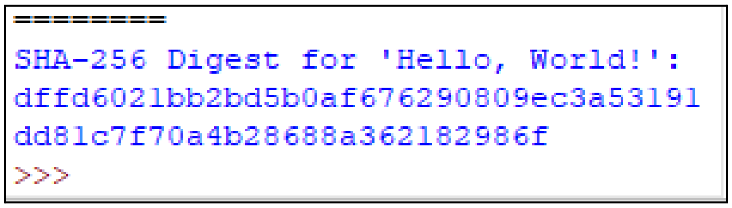
b. SHA Message Digest

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.

Source Code:

```
import hashlib

def sha256_digest(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

# Example usage
message = "Hello, World!"
result = sha256_digest(message)
print(f"SHA-256 Digest for '{message}': {result}")
```

Output:



```
========
SHA-256 Digest for 'Hello, World!':
dffd6021bb2bd5b0af676290809ec3a53191
dd81c7f70a4b28688a362182986f
>>>
```

3. Write the following programs for Blockchain in Python

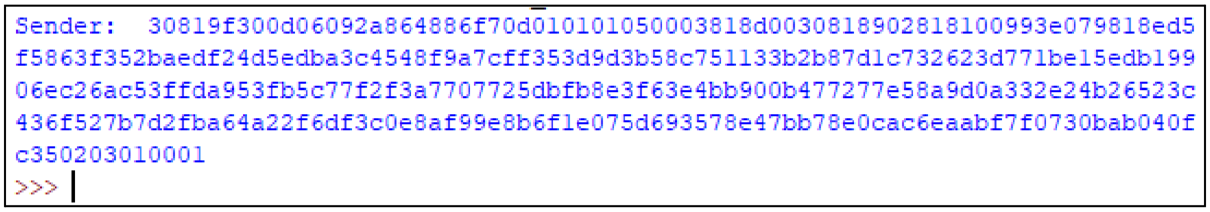
a. A simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it

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.

Source Code:

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

class Client:
    def __init__(self):
        random_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')

Dinesh = Client()
print("Sender: ", Dinesh.identity)
```

Output:



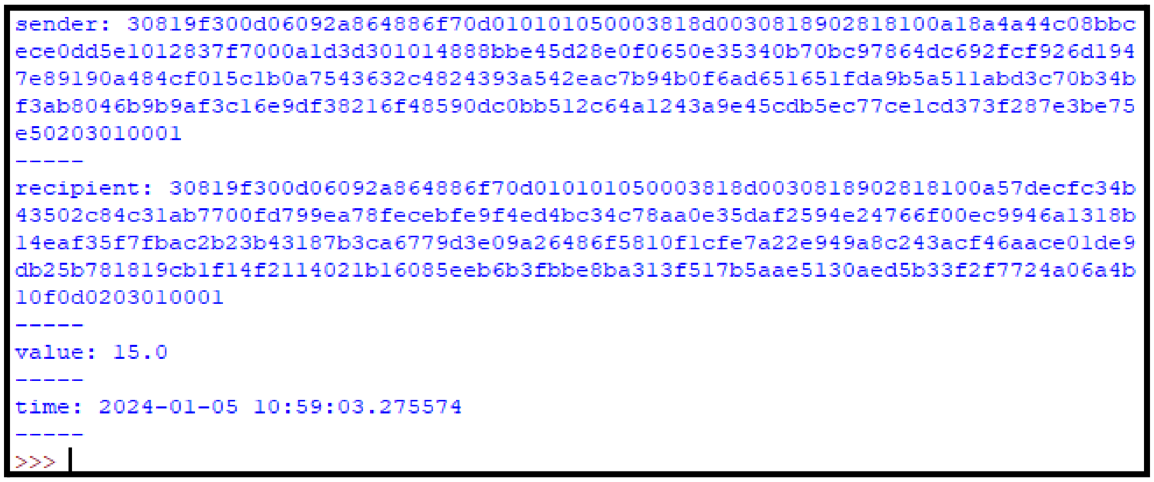
```
Sender:  30819f300d06092a864886f70d010101050003818d0030818902818100993e079818ed5
f5863f352baedf24d5edba3c4548f9a7cff353d9d3b58c751133b2b87d1c732623d771be15edb199
06ec26ac53ffda953fb5c77f2f3a7707725dbfb8e3f63e4bb900b477277e58a9d0a332e24b26523c
436f527b7d2fba64a22f6df3c0e8af99e8b6f1e075d693578e47bb78e0cac6eaabf7f0730bab040f
c350203010001
>>> 
```

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

Source Code:

```
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('-----')

transactions = []
A = Client()
B = Client()

t1 = Transaction(A, B.identity, 15.0)
t1.sign_transaction()

display_transaction(t1)
```

Output:



```
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a18a4a44c08bbc
ece0dd5e1012837f7000a1d3d301014888bbe45d28e0f0650e35340b70bc97864dc692fcf926d194
7e89190a484cf015c1b0a7543632c4824393a542eac7b94b0f6ad651651fda9b5a511abd3c70b34b
f3ab8046b9b9af3c16e9df38216f48590dc0bb512c64a1243a9e45cdb5ec77ce1cd373f287e3be75
e50203010001
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a57decfc34b
43502c84c31ab7700fd799ea78fecebfe9f4ed4bc34c78aa0e35daf2594e24766f00ec9946a1318b
14eaf35f7fbac2b23b43187b3ca6779d3e09a26486f5810f1cfe7a22e949a8c243acf46aace01de9
db25b781819cb1f14f2114021b16085eeb6b3fbbe8ba313f517b5aae5130aed5b33f2f7724a06a4b
10f0d0203010001
-----
value: 15.0
-----
time: 2024-01-05 10:59:03.275574
-----
>>> 
```

4. Write the following programs for Blockchain in Python

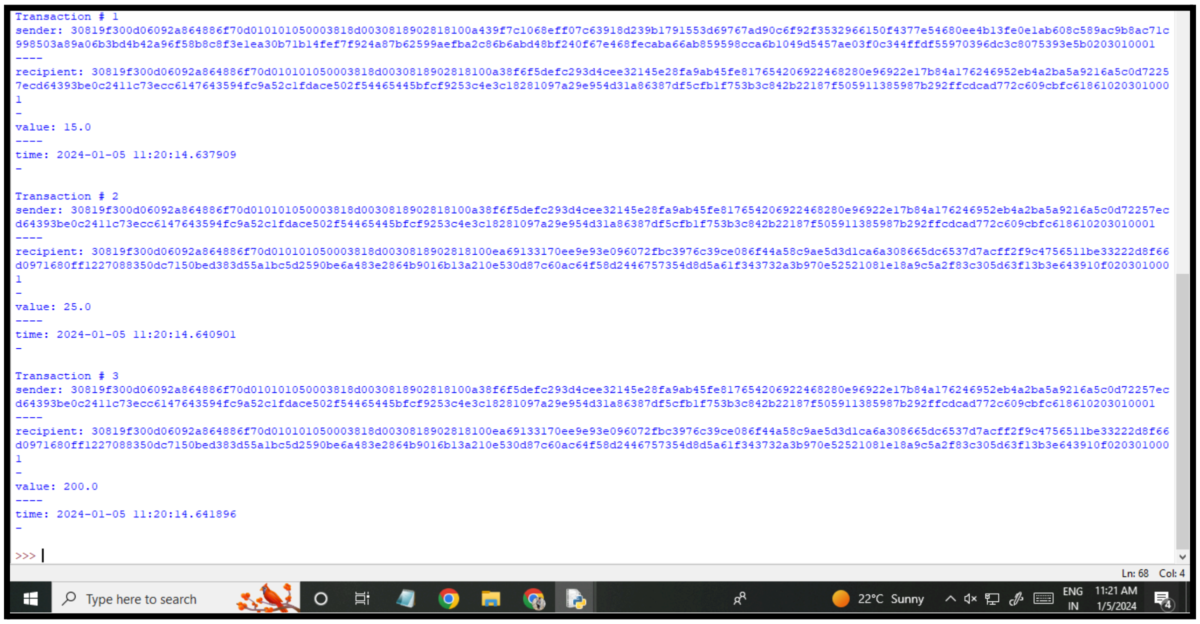
a. Create multiple transactions and display them .

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.

Source Code:

```
import hashlib
import binascii
import datetime
import collections

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

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

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

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

    def to_dict(self):
        if self.sender == "Genesis":
            identity = "Genesis"
        else:
            identity = self.sender.identity
        return collections.OrderedDict({
            'sender': identity,
            'recipient': self.recipient,
            'value': self.value,
            'time': self.time
        })

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

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

transactions = []
Dinesh = Client()
Ramesh = Client()
Suresh = Client()

t1 = Transaction(Dinesh, Ramesh.identity, 15.0)
t1.sign_transaction()
transactions.append(t1)

t2 = Transaction(Ramesh, Suresh.identity, 25.0)
t2.sign_transaction()
transactions.append(t2)

t3 = Transaction(Ramesh, Suresh.identity, 200.0)
t3.sign_transaction()
transactions.append(t3)

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

Output:



```
Transaction # 1
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a439f7c1068eff07c63918d239b1791553d69767ad90c6f92f3532966150f4377e54680ee4b13fe0e1ab608c589ac9b8ac71c
998503a89a06b3bd4b42a96f58b8c8f3e1ea30b71b14fef7f924a87b62599aefba2c86b6abd48bf240f67e468fecaba66ab859598cca6b1049d5457ae03f0c344ffdf55970396dc3c8075393e5b0203010001
----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a38f6f5defc293d4cee32145e28fa9ab45fe817654206922468280e96922e17b84a176246952eb4a2ba5a9216a5c0d7225
7ecd64393be0c2411c73ecc6147643594fc9a52c1fdace502f54465445bfcf9253c4e3c18281097a29e954d31a86387df5cfb1f753b3c842b22187f505911385987b292ffcdcad772c609cbfc61861020301000
1
-
value: 15.0
----
time: 2024-01-05 11:20:14.637909
-

Transaction # 2
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a38f6f5defc293d4cee32145e28fa9ab45fe817654206922468280e96922e17b84a176246952eb4a2ba5a9216a5c0d72257ec
d64393be0c2411c73ecc6147643594fc9a52c1fdace502f54465445bfcf9253c4e3c18281097a29e954d31a86387df5cfb1f753b3c842b22187f505911385987b292ffcdcad772c609cbfc618610203010001
----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100ea69133170ee9e93e096072fbc3976c39ce086f44a58c9ae5d3d1ca6a308665dc6537d7acff2f9c4756511be33222d8f66
d0971680ff1227088350dc7150bed383d55a1bc5d2590be6a483e2864b9016b13a210e530d87c60ac64f58d2446757354d8d5a61f343732a3b970e52521081e18a9c5a2f83c305d63f13b3e643910f020301000
1
-
value: 25.0
----
time: 2024-01-05 11:20:14.640901
-

Transaction # 3
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a38f6f5defc293d4cee32145e28fa9ab45fe817654206922468280e96922e17b84a176246952eb4a2ba5a9216a5c0d72257ec
d64393be0c2411c73ecc6147643594fc9a52c1fdace502f54465445bfcf9253c4e3c18281097a29e954d31a86387df5cfb1f753b3c842b22187f505911385987b292ffcdcad772c609cbfc618610203010001
----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100ea69133170ee9e93e096072fbc3976c39ce086f44a58c9ae5d3d1ca6a308665dc6537d7acff2f9c4756511be33222d8f66
d0971680ff1227088350dc7150bed383d55a1bc5d2590be6a483e2864b9016b13a210e530d87c60ac64f58d2446757354d8d5a61f343732a3b970e52521081e18a9c5a2f83c305d63f13b3e643910f020301000
1
-
value: 200.0
----
time: 2024-01-05 11:20:14.641896
-

>>> |
```

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

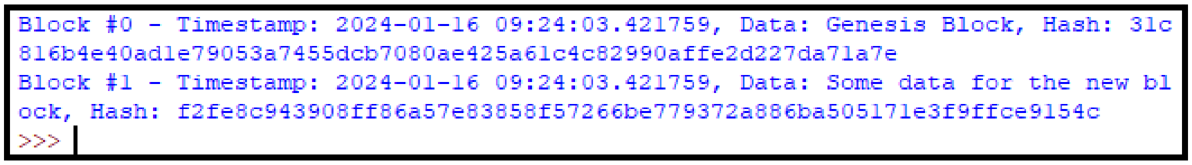
The term "genesis block" refers to the inaugural block in a blockchain. It serves as the starting point of the entire blockchain network and contains no reference to a previous block. The creation of the genesis block marks the initiation of the blockchain, and subsequent blocks build upon it as the 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.

Source Code:

```
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"Block #{block.index} - Timestamp: {block.timestamp}, Data: {block.data},
Hash: {block.hash}")
```

Output:



```
Block #0 - Timestamp: 2024-01-16 09:24:03.421759, Data: Genesis Block, Hash: 31c
816b4e40ad1e79053a7455dcb7080ae425a61c4c82990affe2d227da71a7e
Block #1 - Timestamp: 2024-01-16 09:24:03.421759, Data: Some data for the new bl
ock, Hash: f2fe8c943908ff86a57e83858f57266be779372a886ba505171e3f9ffce9154c
>>> 
```

5. Write the following programs for Blockchain in Python

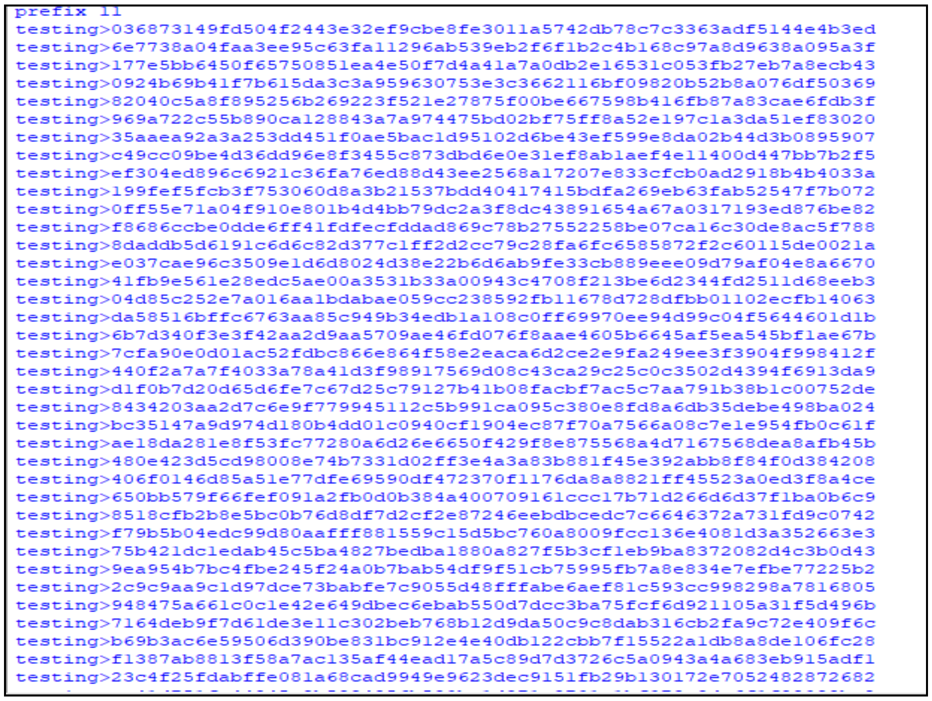
a. Create a mining function and test it.

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.

Source Code:

```
import hashlib
def sha256(message):
    return hashlib.sha256(message.encode('ascii')).hexdigest()
def mine(message, difficulty=1):
    assert difficulty>= 1
    #if(difficulty <1):
    # return
    #1*2=> '11'
    prefix = '1' * difficulty
    print("prefix",prefix)
    for i in range(1000):
        digest = sha256(str(hash(message)) + str(i))
        print("testing>"+digest)
        if digest.startswith(prefix):
            print ("after" + str(i) + " iterations found nonce: "+ digest)
            return i #i nonce value

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

Output:



```
prefix 11
testing>036873149fd504f2443e32ef9cbe8fe3011a5742db78c7c3363adf5144e4b3ed
testing>6e7738a04faa3ee95c63fa11296ab539eb2f6f1b2c4b168c97a8d9638a095a3f
testing>177e5bb6450f65750851ea4e50f7d4a41a7a0db2e16531c053fb27eb7a8ecb43
testing>0924b69b41f7b615da3c3a959630753e3c3662116bf09820b52b8a076df50369
testing>82040c5a8f895256b269223f521e27875f00be667598b416fb87a83cae6fdb3f
testing>969a722c55b890ca128843a7a974475bd02bf75ff8a52e197c1a3da51ef83020
testing>35aaea92a3a253dd451f0ae5bac1d95102d6be43ef599e8da02b44d3b0895907
testing>c49cc09be4d36dd96e8f3455c873dbd6e0e31ef8ab1aef4e11400d447bb7b2f5
testing>ef304ed896c6921c36fa76ed88d43ee2568a17207e833cfcb0ad2918b4b4033a
testing>199fef5fcb3f753060d8a3b21537bdd40417415bdfa269eb63fab52547f7b072
testing>0ff55e71a04f910e801b4d4bb79dc2a3f8dc43891654a67a0317193ed876be82
testing>f8686ccbe0dde6ff41fdfecfddad869c78b27552258be07ca16c30de8ac5f788
testing>8daddb5d6191c6d6c82d377c1ff2d2cc79c28fa6fc6585872f2c60115de0021a
testing>e037cae96c3509e1d6d8024d38e22b6d6ab9fe33cb889eee09d79af04e8a6670
testing>41fb9e561e28edc5ae00a3531b33a00943c4708f213be6d2344fd2511d68eeb3
testing>04d85c252e7a016aa1bdabae059cc238592fb11678d728dfbb01102ecfb14063
testing>da58516bffc6763aa85c949b34edb1a108c0ff69970ee94d99c04f5644601d1b
testing>6b7d340f3e3f42aa2d9aa5709ae46fd076f8aae4605b6645af5ea545bf1ae67b
testing>7cfa90e0d01ac52fdbc866e864f58e2eaca6d2ce2e9fa249ee3f3904f998412f
testing>440f2a7a7f4033a78a41d3f98917569d08c43ca29c25c0c3502d4394f6913da9
testing>d1f0b7d20d65d6fe7c67d25c79127b41b08facbf7ac5c7aa791b38b1c00752de
testing>8434203aa2d7c6e9f779945112c5b991ca095c380e8fd8a6db35debe498ba024
testing>bc35147a9d974d180b4dd01c0940cf1904ec87f70a7566a08c7e1e954fb0c61f
testing>ae18da281e8f53fc77280a6d26e6650f429f8e875568a4d7167568dea8afb45b
testing>480e423d5cd98008e74b7331d02ff3e4a3a83b881f45e392abb8f84f0d384208
testing>406f0146d85a51e77dfe69590df472370f1176da8a8821ff45523a0ed3f8a4ce
testing>650bb579f66fef091a2fb0d0b384a400709161ccc17b71d266d6d37f1ba0b6c9
testing>8518cfb2b8e5bc0b76d8df7d2cf2e87246eebdbcedc7c6646372a731fd9c0742
testing>f79b5b04edc99d80aafff881559c15d5bc760a8009fcc136e4081d3a352663e3
testing>75b421dc1edab45c5ba4827bedba1880a827f5b3cf1eb9ba8372082d4c3b0d43
testing>9ea954b7bc4fbe245f24a0b7bab54df9f51cb75995fb7a8e834e7efbe77225b2
testing>2c9c9aa9c1d97dce73babfe7c9055d48fffabe6aef81c593cc998298a7816805
testing>948475a661c0c1e42e649dbec6ebab550d7dcc3ba75fcf6d921105a31f5d496b
testing>7164deb9f7d61de3e11c302beb768b12d9da50c9c8dab316cb2fa9c72e409f6c
testing>b69b3ac6e59506d390be831bc912e4e40db122cbb7f15522a1db8a8de106fc28
testing>f1387ab8813f58a7ac135af44ead17a5c89d7d3726c5a0943a4a683eb915adf1
testing>23c4f25fdabffe081a68cad9949e9623dec9151fb29b130172e7052482872682
```

6. Write the solidity Code and demonstrate the following:

a. Variable

1. Local Variable

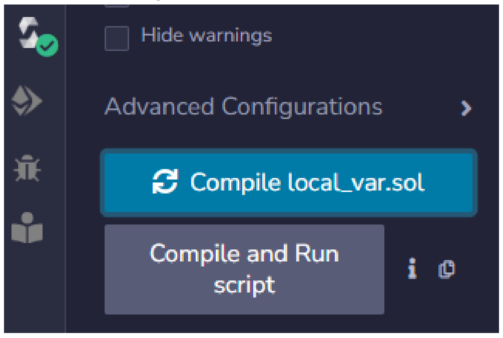
   Local variables are declared within a specific function or code block and exist only for the duration of that particular execution. They are used for temporary storage and calculations within the confines of a single function.

   Source Code:

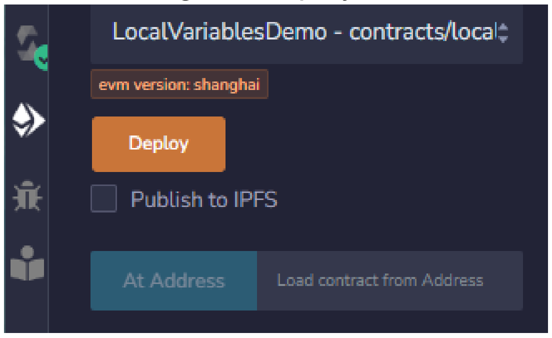
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract LocalVariablesDemo {

    function calculateSum(uint256 a, uint256 b) public pure returns (uint256) {
        uint256 result;
        result = a + b;
        return result;
    }
}
```

   Output:

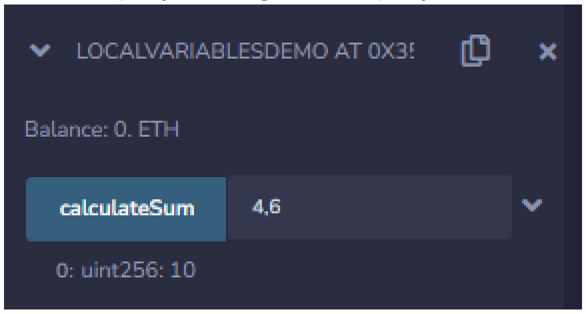
   Go to Compiler



   Now go to Deploy & Run


LocalVariablesDemo - contracts/local
evm version: shanghai
Deploy
Publish to IPFS
At Address
Load contract from Address

   After Deployment go to Deployed Contract



```
Balance: 0. ETH
calculateSum 4,6
0: uint256: 10
```

2. State Variable

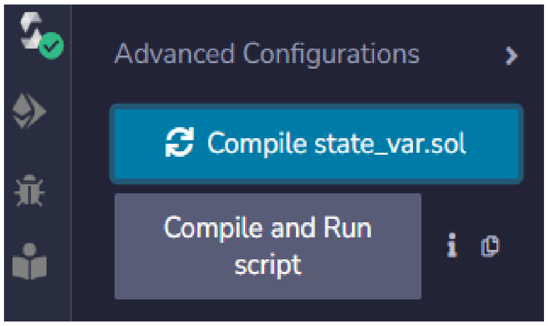
   State variables, on the other hand, are declared outside any function within the contract and have a more extended lifetime. They persist across multiple function calls and endure as long as the contract is deployed.

   Source Code:

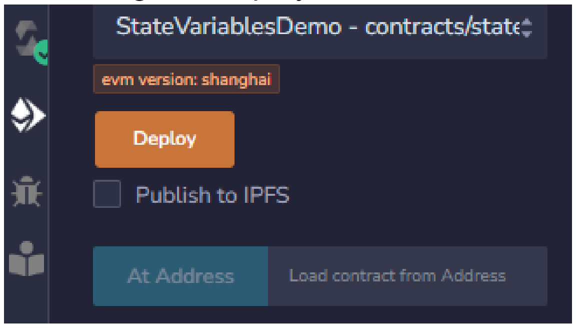
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract StateVariablesDemo {

    uint256 public myNumber;

    function setNumber(uint256 newValue) public {
        myNumber = newValue;
    }
}
```

Output:

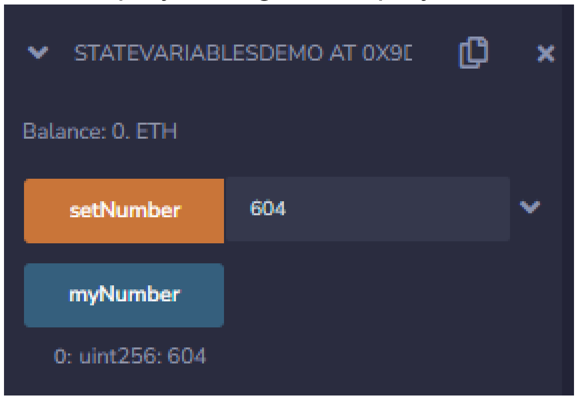
Go to Compiler



Now go to Deploy & Run


StateVariablesDemo - contracts/state
evm version: shanghai
Deploy
Publish to IPFS
At Address
Load contract from Address

After Deployment go to Deployed Contract



Balance: 0. ETH

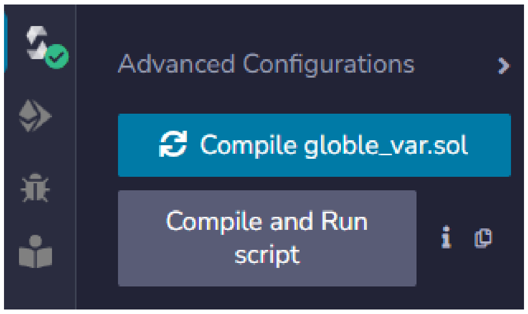
604

0: uint256: 604

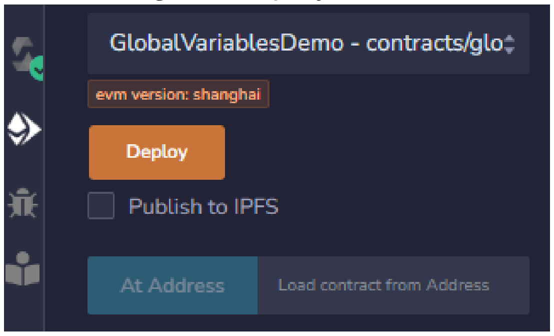
3. Global Variable
Function parameters are variables declared within a function's signature, serving as inputs to the function. Similar to local variables, function parameters have a limited scope and lifetime, existing only for the duration of the function's execution. They allow external values to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

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

contract GlobalVariablesDemo {
    uint256 public myGlobalNumber;
    constructor() {
        myGlobalNumber = 0;
    }

    function updateGlobalNumber(uint256 newValue) public {
        myGlobalNumber = newValue;
    }
}
```

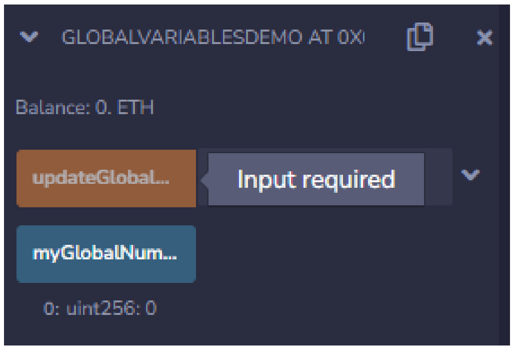
Output:
Go to Compiler



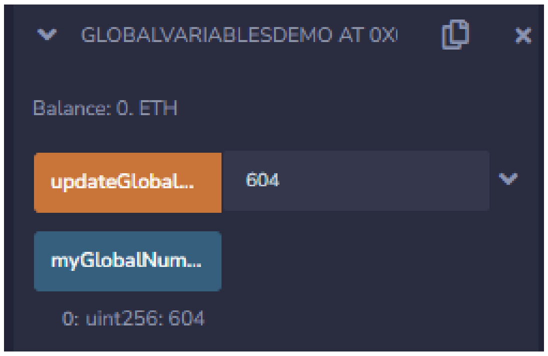
Now go to Deploy & Run


GlobalVariablesDemo - contracts/glo
evm version: shanghai

After Deployment go to Deployed Contract



Balance: 0. ETH
0: uint256: 0



Balance: 0. ETH
604
0: uint256: 604

b. Operators

Solidity, Ethereum's smart contract language, features operators for arithmetic (e.g., +, -, *, /), comparisons (==, !=, <, >), logical (&&, ||, !), bitwise (&, |, ^, ~, <<, >>), and assignment (=, +=, -=). These operators empower developers to perform various computations, comparisons, and bit-level operations efficiently within smart contracts.

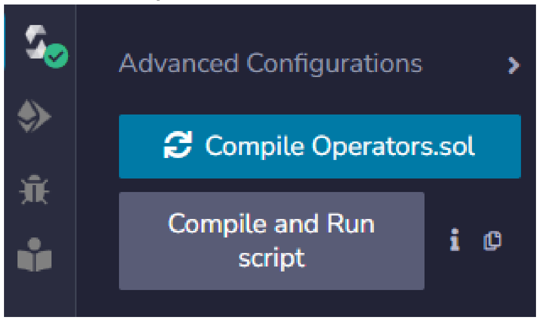
Source Code:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.12 <0.9.0;
contract operators {
    // Arithmetic operators
    uint public additionResult = 5 + 3;
    uint public subtractionResult = 10 - 4;
```

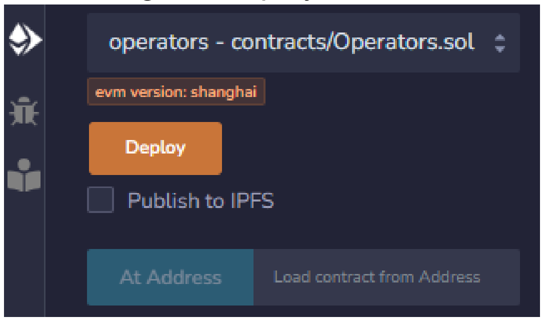
```
        uint public multiplicationResult = 6 * 2;
        uint public divisionResult = 16 / 2;
        uint public moduloResult = 15 % 4;

        // Comparison operators
        bool public isEqual = (7 == 7);
        bool public isNotEqual = (5 != 3);
        bool public isGreaterThan = (10 > 5);
        bool public isLessThan = (3 < 8);
        bool public isGreaterOrEqual = (6 >= 6);
        bool public isLessOrEqual = (9 <= 10);

        // Logical operators
        bool public andOperator = true && false;
        bool public orOperator = true || false;
        bool public notOperator = !true;
    }
```

Output:

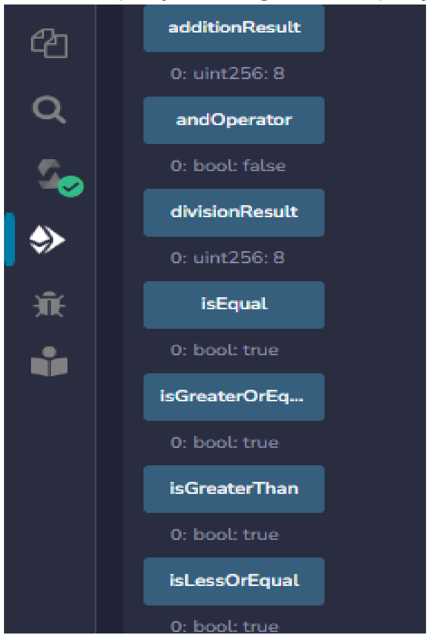
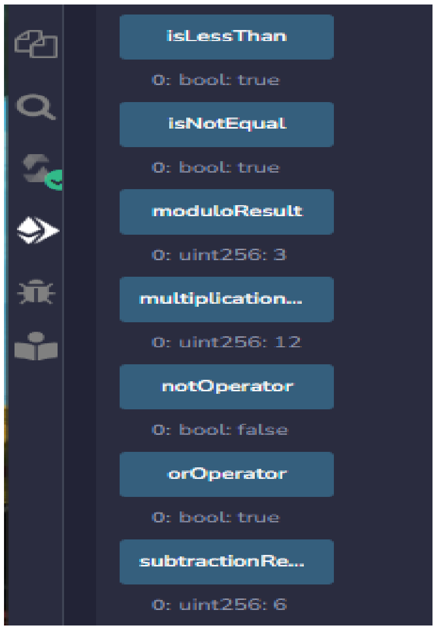
Go to Compiler



Now go to Deploy & Run


operators - contracts/Operators.sol
evm version: shanghai
Deploy
Publish to IPFS
At Address
Load contract from Address

After Deployment go to Deployed Contract

```
additionResult
0: uint256: 8
andOperator
0: bool: false
divisionResult
0: uint256: 8
isEqual
0: bool: true
isGreaterOrEq...
0: bool: true
isGreaterThan
0: bool: true
isLessOrEqual
0: bool: true
```

```
isLessThan
0: bool: true
isNotEqual
0: bool: true
moduloResult
0: uint256: 3
multiplication...
0: uint256: 12
notOperator
0: bool: false
orOperator
0: bool: true
subtractionRe...
0: uint256: 6
```

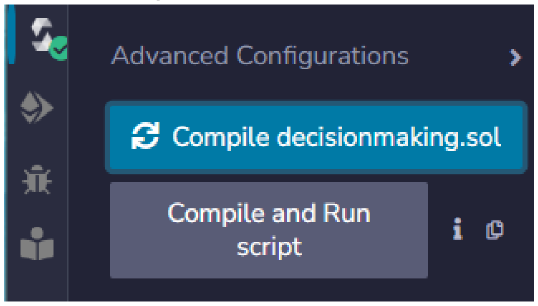
c. Decision Making - if, if else, if else if

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.

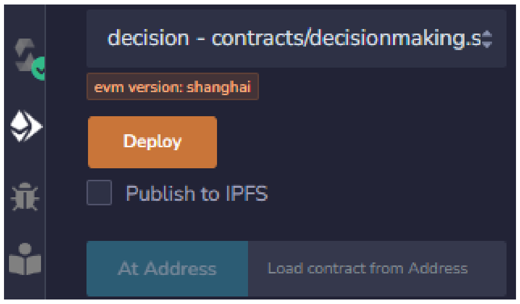
Source Code:

```
// 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.";
        }
    }
}
```

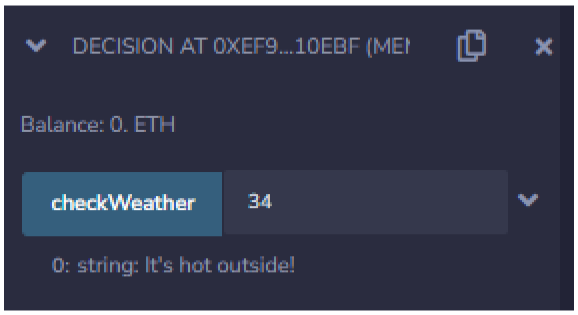
Go to Compiler



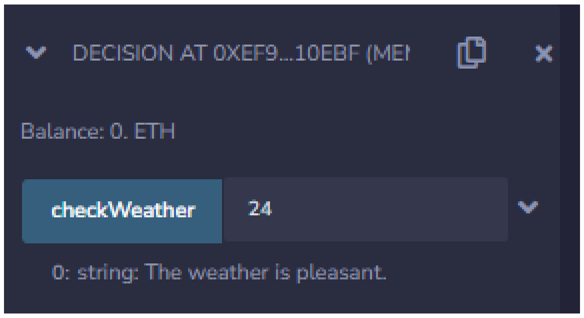
Now go to Deploy & Run


decision - contracts/decisionmaking.s
evm version: shanghai
Deploy
Publish to IPFS
At Address
Load contract from Address

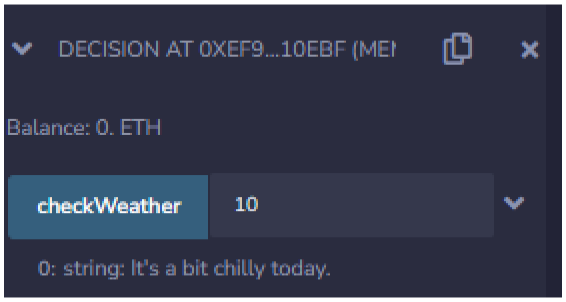
After Deployment go to Deployed Contract



```
0: string: It's hot outside!
```



```
0: string: The weather is pleasant.
```



0: string: It's a bit chilly today.

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.

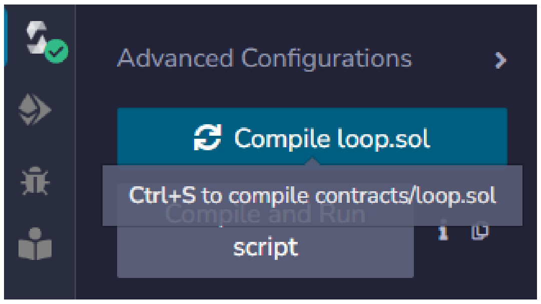
Source Code:

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

   uint public sum;
   uint public whileLoopSum;
   uint public doWhileLoopSum;

   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 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;
   }
```

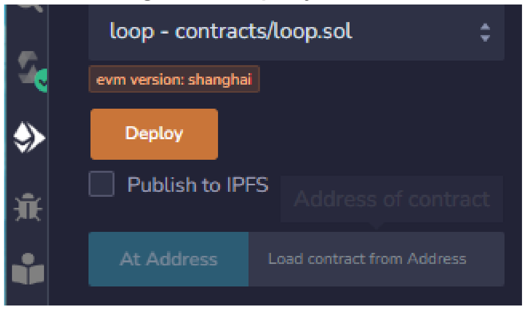
```
    function calculateDoWhileLoopSum(uint n) public returns(uint) {
        // Initialize variables
        uint i = 1;
        doWhileLoopSum = 0;

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

Output:

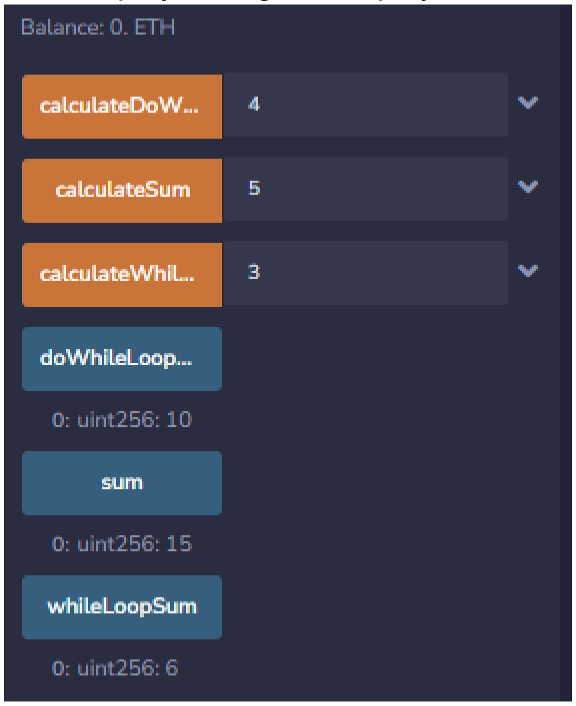
Go to Compiler



Now go to Deploy & Run



After Deployment go to Deployed Contract



```
0: uint256: 10
0: uint256: 15
0: uint256: 6
```

7. Write the solidity Code and demonstrate the following

a. Functions

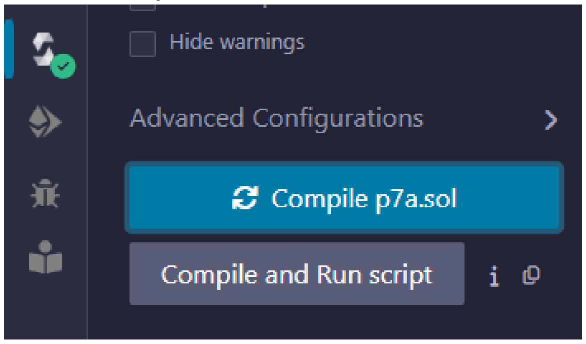
Functions in Solidity serve as modular units of code, allowing smart contracts to execute specific tasks, manage data, and interact with the blockchain. They enhance the organization and readability of contract logic, supporting both internal and external calls.

Source Code:

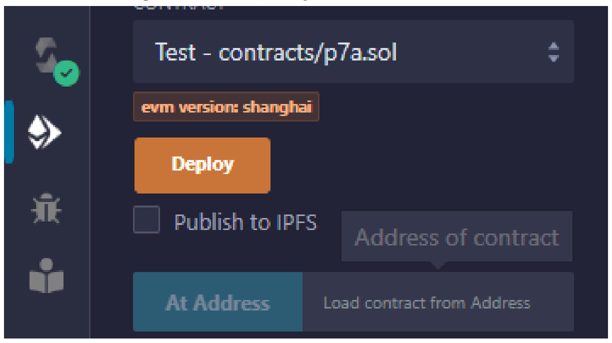
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

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

Output:

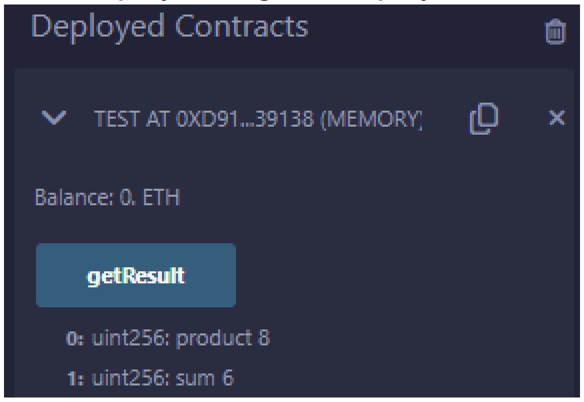
Go to Compiler



Now go to Deploy & Run



After Deployment go to Deployed Contract



```
Balance: 0. ETH
0: uint256: product 8
1: uint256: sum 6
```

b. View Functions

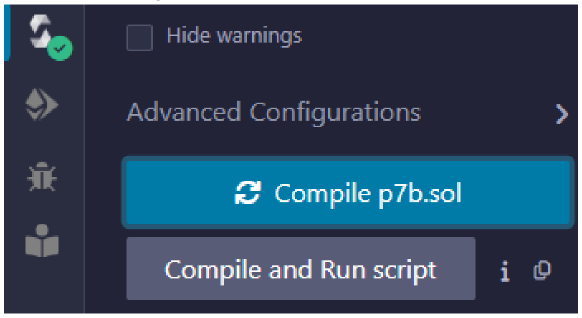
View functions in Solidity provide a read-only perspective, refraining from modifying the contract's state. Primarily used for data retrieval, they execute locally on the caller's machine, minimizing gas costs and facilitating efficient interactions with the contract.

Source Code:

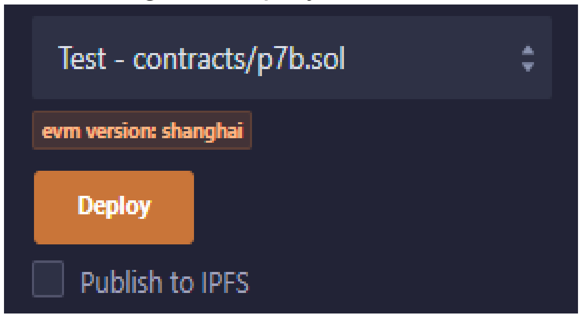
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

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

Output:

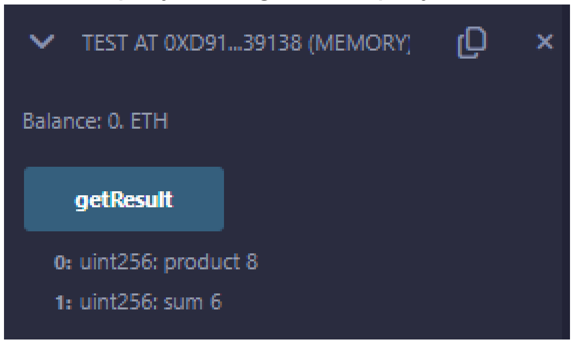
Go to Compiler



Now go to Deploy & Run



After Deployment go to Deployed Contract



```
0: uint256: product 8
1: uint256: sum 6
```

c. Pure Functions

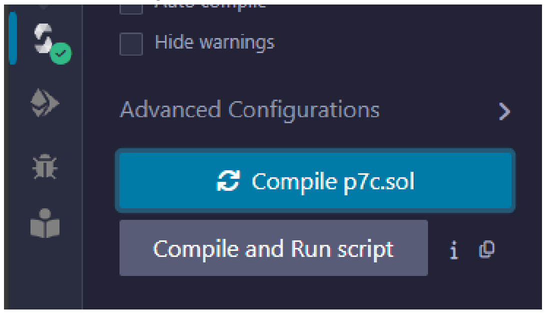
Pure functions extend the concept of non-state modification by not accessing the contract's state at all. Relying solely on input parameters, they guarantee deterministic outcomes, making them suitable for mathematical calculations and utility functions that demand predictability and security.

Source Code:

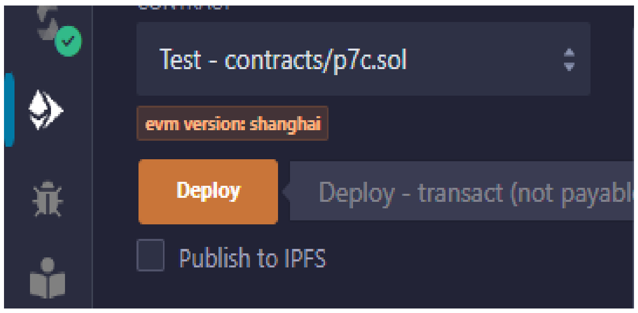
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Test {
   function getResult() public pure returns(uint product, uint sum){
   uint num1 = 5;
   uint num2 = 4;
   product=num1*num2;
   sum = num1 + num2;
   }
}
```

Output:

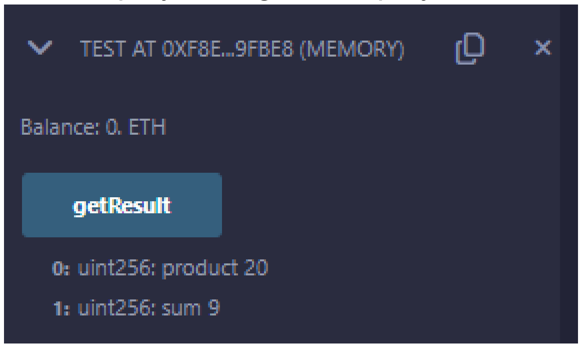
Go to Compiler



Now go to Deploy & Run



After Deployment go to Deployed Contract



```
0: uint256: product 20
1: uint256: sum 9
```

d. Cryptographic Function

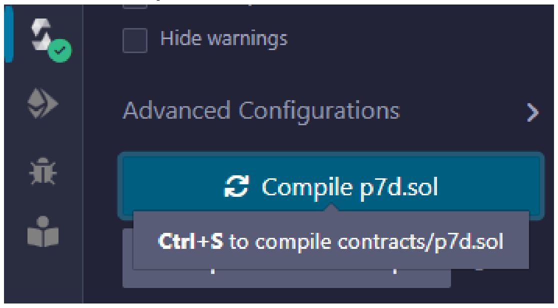
Cryptographic functions in Solidity, like keccak256, facilitate hashing, digital signatures, and message verification, ensuring transaction security, ownership proof, and data integrity in decentralized applications on the Ethereum blockchain. They are crucial for safeguarding sensitive information.

Source Code:

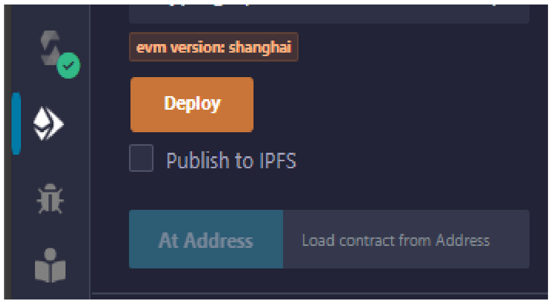
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract CryptographicFunctions {
  function callsha256() public pure returns(bytes32 result){
   return sha256("ronaldino");
  }
  function callkeccak256() public pure returns(bytes32 result){
   return  keccak256("ronaldino");
  }
}
```

Output:

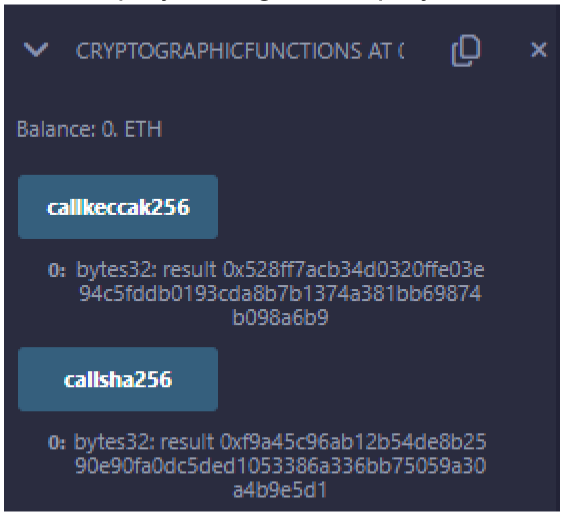
Go to Compiler


Hide warnings
Advanced Configurations
Compile p7d.sol
Ctrl+S to compile contracts/p7d.sol

Now go to Deploy & Run


evm version: shanghai
Deploy
Publish to IPFS
At Address
Load contract from Address

After Deployment go to Deployed Contract


CRYPTOGRAPHICFUNCTIONS AT (

Balance: 0. ETH

callkeccak256

```
0: bytes32: result 0x528ff7acb34d0320ffe03e
94c5fddb0193cda8b7b1374a381bb69874
b098a6b9
```

callsha256

```
0: bytes32: result 0xf9a45c96ab12b54de8b25
90e90fa0dc5ded1053386a336bb75059a30
a4b9e5d1
```

8. Write the solidity Code and demonstrate the following

a. Contracts

Contract in Solidity is similar to a Class in C++.

A Contract have following properties.

Constructor − A special function declared with constructor keyword which will be executed once per contract and is invoked when a contract is created.

State Variables − Variables per Contract to store the state of the contract.

Functions − Functions per Contract which can modify the state variables to alter the state of a contract.

b. Inheritance

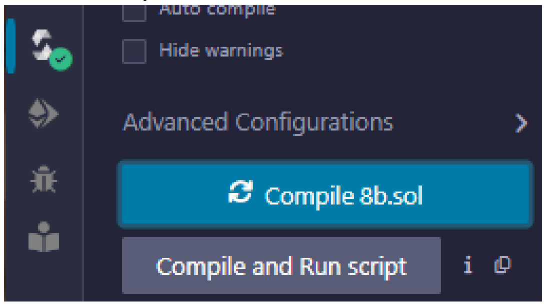
Source Code:

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

contract C {
   // Private state variable
   uint private data;

   // Public state variable
   uint public info;

   // 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;
   }
}
```

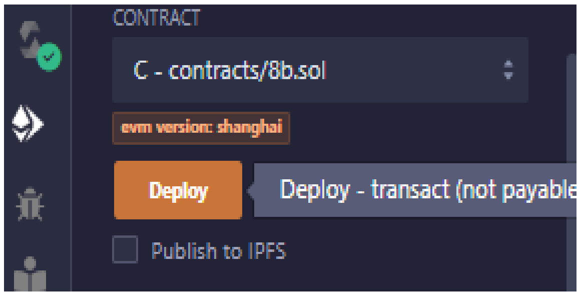
```
// Derived Contract
contract E is C {
    uint private result;
    C private c;

    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();
    }
}
```

Output:

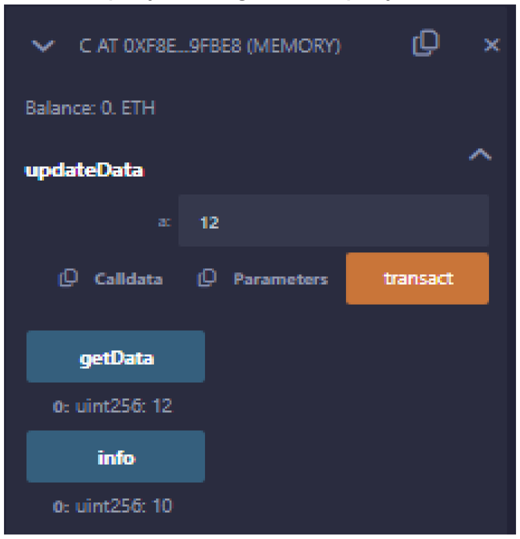
Go to Compiler



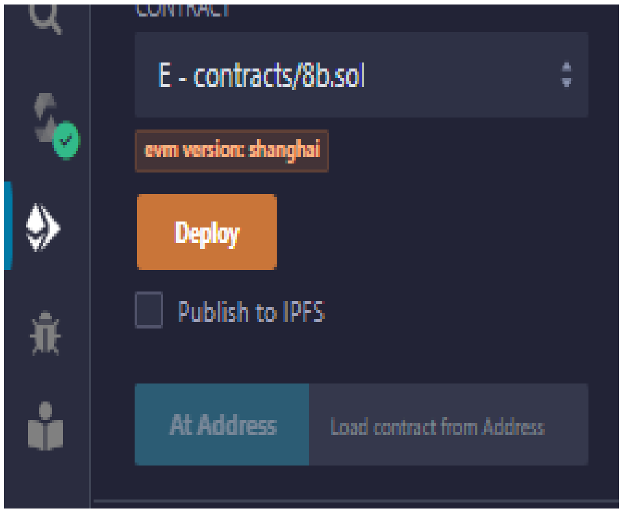
Now go to Deploy & Run



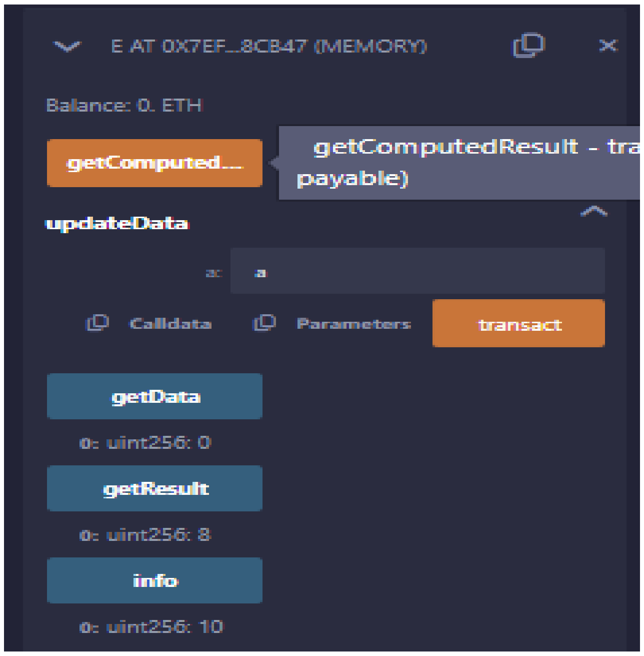
After Deployment go to Deployed Contract



Now go to Deploy & Run



After Deployment go to Deployed Contract



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.

Interface can not have any function with implementation.

Functions of an interface can be only of type external.

Interface can not have constructor.

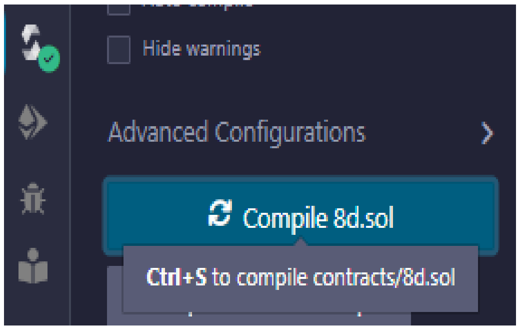
Interface can not have state variables.

Source Code:

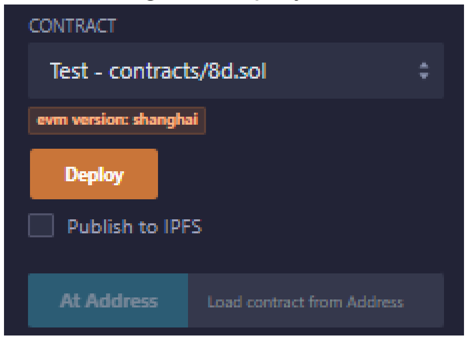
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

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

contract Test is Calculator {
    constructor() {}

    function getResult() external pure returns(uint){
        uint a = 5;
        uint b = 2;
        uint result = a + b;
        return result;
    }
}
```

Output:

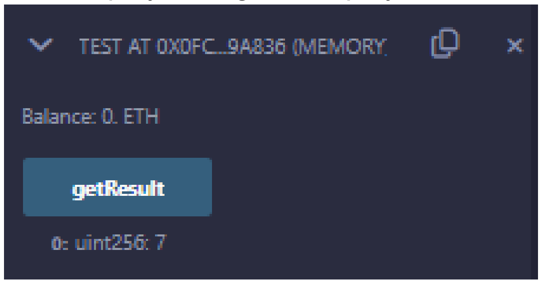
Go to Compiler



Now go to Deploy & Run



After Deployment go to Deployed Contract



```
0: uint256: 7
```

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

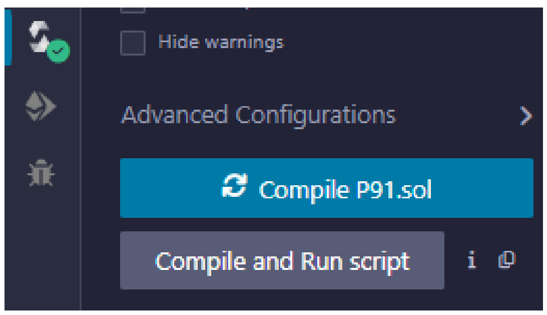
The `Counter` contract allows you to get, increment, and decrement a stored count. The `getCount` function retrieves the count, while `increment` and `decrement` adjust it by 1. The contract emits a `CountUpdated` event for transparency.

Source Code:

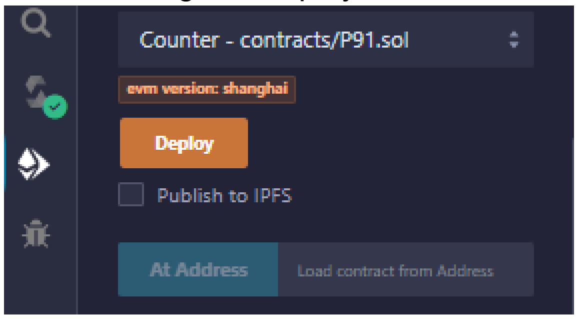
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Counter {
    uint public count;

    // Function to get the current count
    function get() public view returns (uint) {
        return count;
    }

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

    // Function to decrement count by 1
    function dec() public {
        // This function will fail if count = 0
        count -= 1;
    }
}
```

Output:

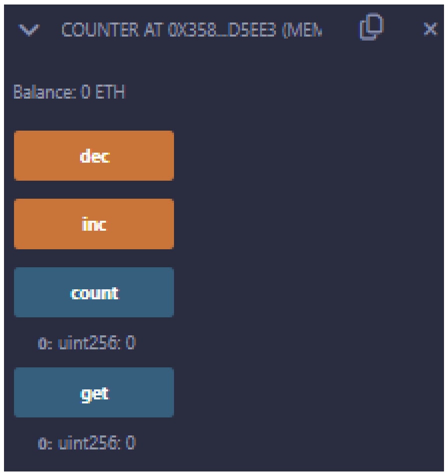
Go to Compiler



Now go to Deploy & Run

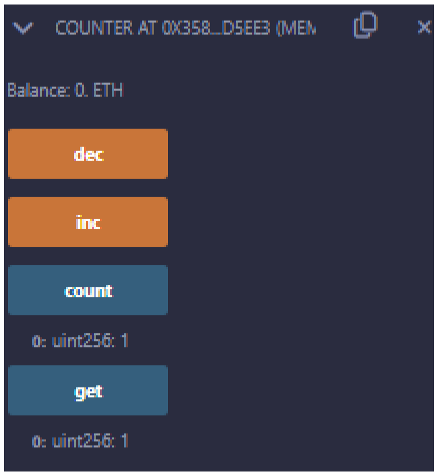


After Deployment go to Deployed Contract



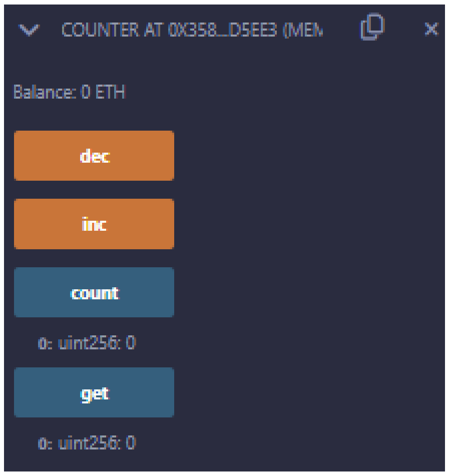
```
COUNTER AT 0X358...D5EE3 (MEM
Balance: 0 ETH
dec
inc
count
0: uint256: 0
get
0: uint256: 0
```

Increment



```
COUNTER AT 0X358...D5EE3 (MEM
Balance: 0. ETH
dec
inc
count
0: uint256: 1
get
0: uint256: 1
```

Decrement



```
COUNTER AT 0X358...D5EE3 (MEM
Balance: 0 ETH
dec
inc
count
0: uint256: 0
get
0: uint256: 0
```

10.

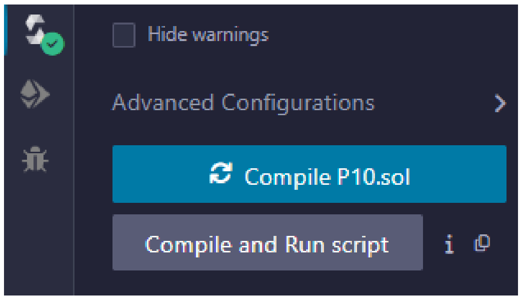
a. Mint Your Own Coins and Make Simple Transactions With Solidity

Source Code:

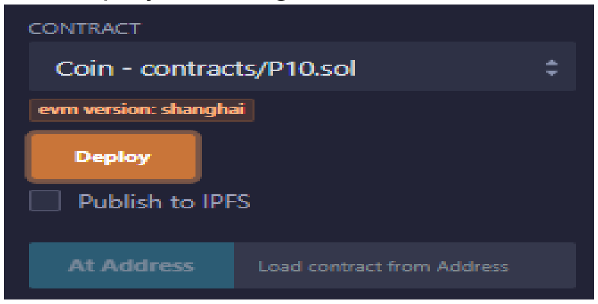
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;
    event Sent(address from, address to, uint amount);
    constructor(){
        minter = msg.sender;
    }

    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);
    }
}
```

Output:

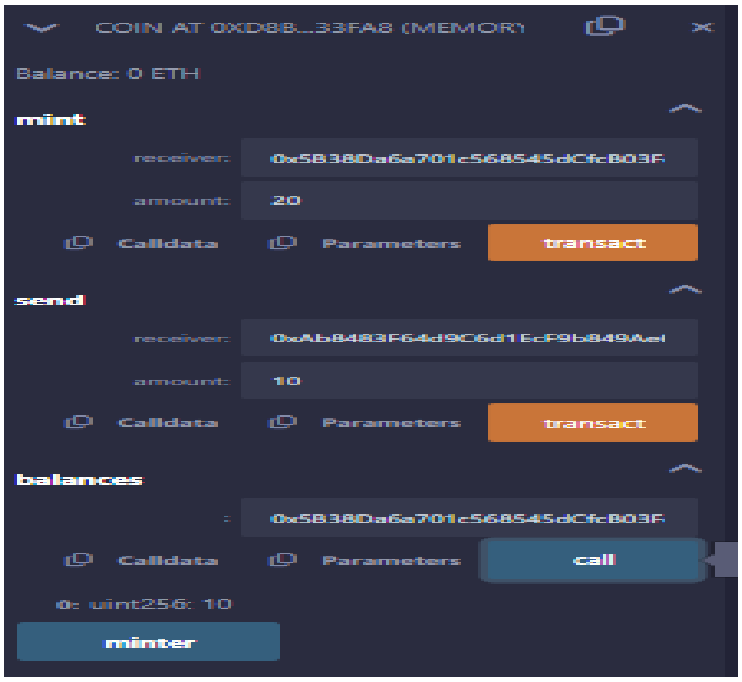
Compile the program



Deploy the Program



Noe copy the address of deployment and paste it into mint, add some amount and transect then check balance. Get the address of other sender deploy and send than check the balance of your own



COIN AT 0XD8B...33FA8 (MEMORY)

Balance: 0 ETH

mint

receiver: 0x5B38Da6a701c568545dCfcB03F

amount: 20

send

receiver: 0xAb8483F64d9C6d1EcF9b849Ae

amount: 10

balances

: 0x5B38Da6a701c568545dCfcB03F

0: uint256: 10

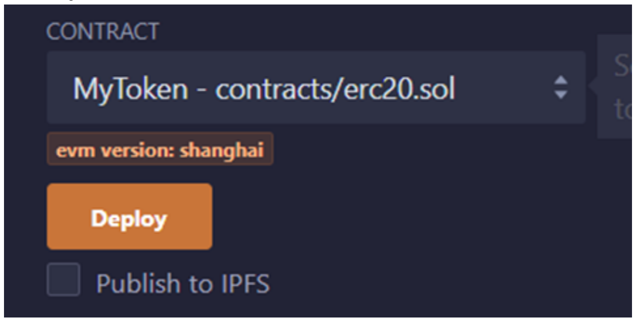
b. Creating ERC -20 Token

Source Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';

contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "MT") {
        _mint(msg.sender, 1000 * (10 ** uint256(decimals())));
    }
}
```

Output:

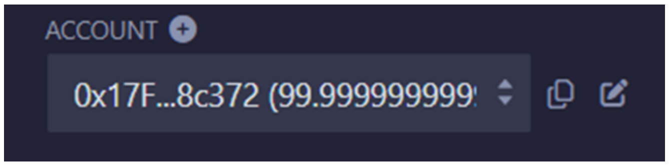
Compile



CONTRACT

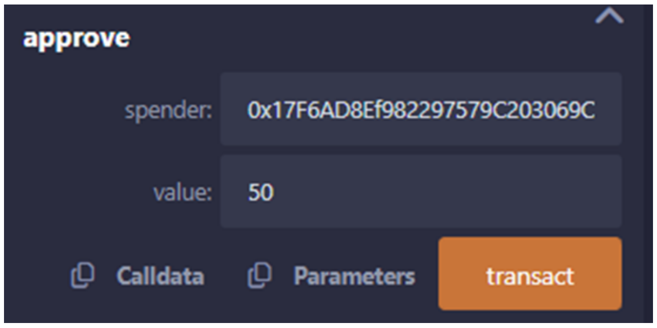
MyToken - contracts/erc20.sol

evm version: shanghai



ACCOUNT

0x17F...8c372 (99.999999999

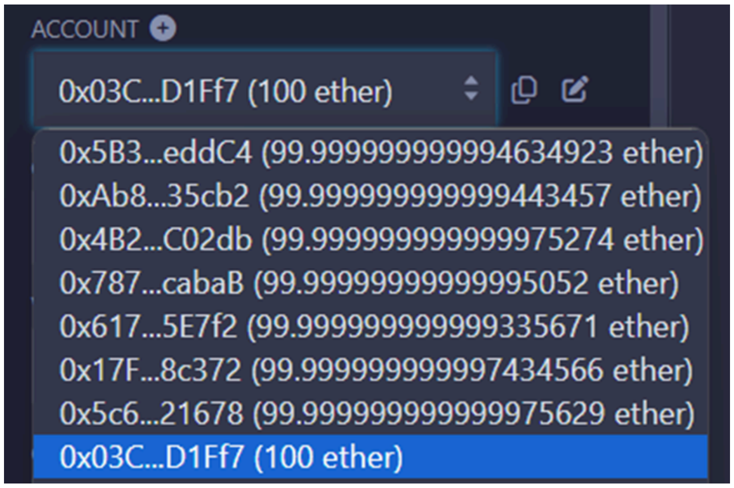


approve

spender: 0x17F6AD8Ef982297579C203069C

value: 50

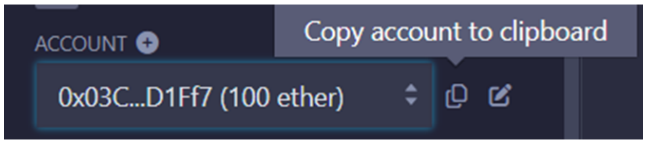
Calldata Parameters transact



ACCOUNT

0x03C...D1Ff7 (100 ether)

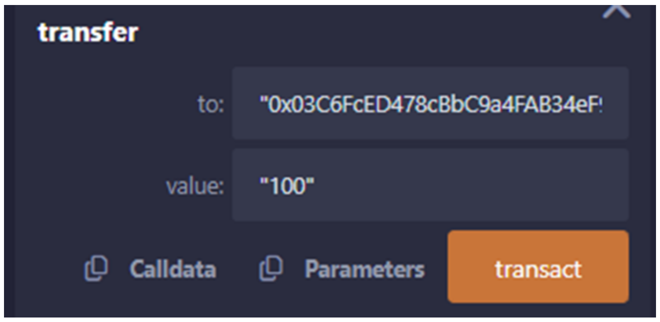
```
0x5B3...eddC4 (99.999999999994634923 ether)
0xAb8...35cb2 (99.999999999999443457 ether)
0x4B2...C02db (99.999999999999975274 ether)
0x787...cabaB (99.99999999999995052 ether)
0x617...5E7f2 (99.999999999999335671 ether)
0x17F...8c372 (99.999999999997434566 ether)
0x5c6...21678 (99.999999999999975629 ether)
0x03C...D1Ff7 (100 ether)
```



ACCOUNT

Copy account to clipboard

0x03C...D1Ff7 (100 ether)

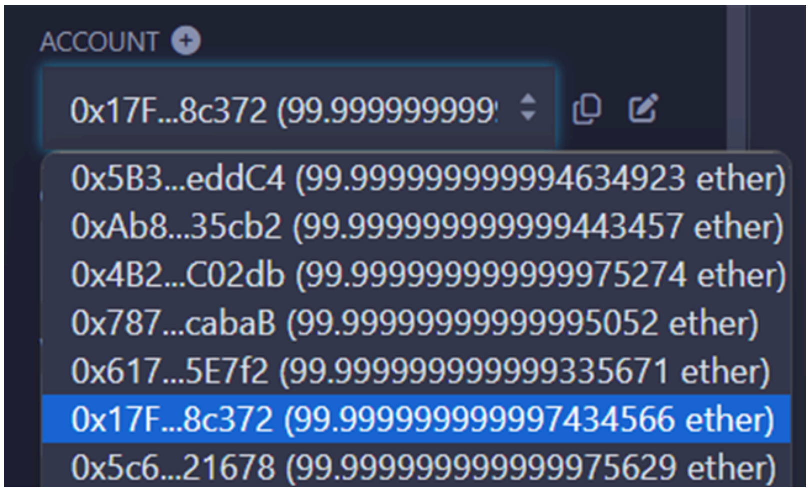


transfer

to: "0x03C6FcED478cBbC9a4FAB34eF!

value: "100"

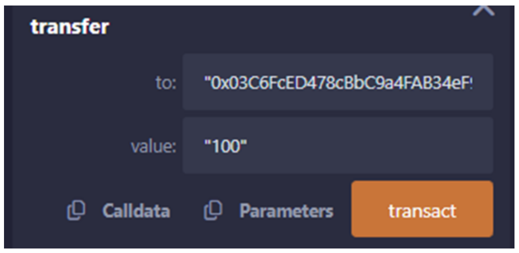
Calldata Parameters transact



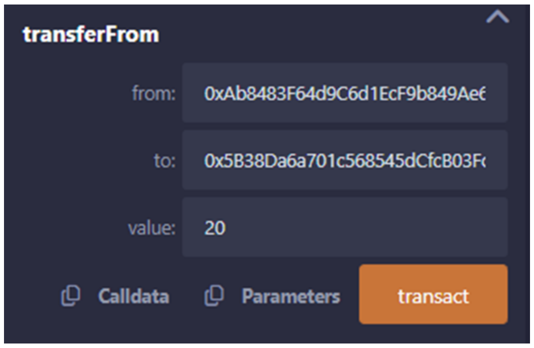
ACCOUNT

0x17F...8c372 (99.999999999!

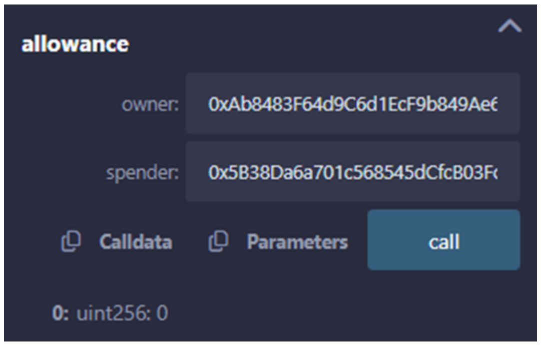
```
0x5B3...eddC4 (99.999999999994634923 ether)
0xAb8...35cb2 (99.999999999999443457 ether)
0x4B2...C02db (99.999999999999975274 ether)
0x787...cabaB (99.99999999999995052 ether)
0x617...5E7f2 (99.999999999999335671 ether)
0x17F...8c372 (99.999999999997434566 ether)
0x5c6...21678 (99.999999999999975629 ether)
```



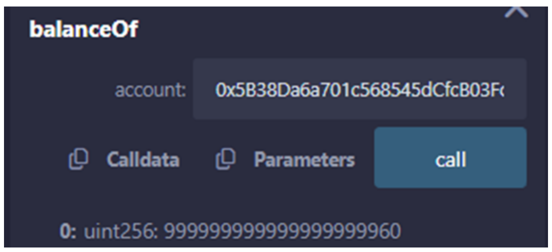
```
transfer
to: "0x03C6FcED478cBbC9a4FAB34eF
value: "100"
```



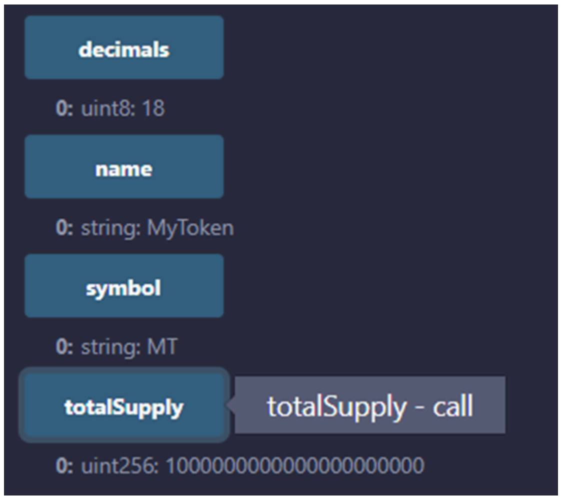
```
transferFrom
from: 0xAb8483F64d9C6d1EcF9b849Ae
to: 0x5B38Da6a701c568545dCfcB03F
value: 20
```



```
allowance
owner: 0xAb8483F64d9C6d1EcF9b849Ae
spender: 0x5B38Da6a701c568545dCfcB03F
0: uint256: 0
```



```
balanceOf
account: 0x5B38Da6a701c568545dCfcB03F
0: uint256: 999999999999999999960
```



```
decimals
0: uint8: 18
name
0: string: MyToken
symbol
0: string: MT
totalSupply
0: uint256: 1000000000000000000000
```