

PRACTICAL 1 A.

Aim: Develop a secure messaging application where users can exchange messages securely using RSA encryption. Implement a mechanism for generating RSA key pairs and encrypting/decrypting messages.

Code:

```
! pip install cryptography
import cryptography
print("Cryptography library is successfully installed!")

from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.serialization import
load_pem_private_key, load_pem_public_key
from cryptography.hazmat.primitives.serialization import Encoding,
PublicFormat, PrivateFormat, NoEncryption

# Generate RSA key pair
def generate_rsa_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    public_key = private_key.public_key()
    return private_key, public_key

# Save keys to PEM format
def save_keys_to_pem(private_key, public_key):
    private_pem = private_key.private_bytes(
        encoding=Encoding.PEM,
        format=PrivateFormat.PKCS8,
        encryption_algorithm=NoEncryption()
    )
    public_pem = public_key.public_bytes(
        encoding=Encoding.PEM,
        format=PublicFormat.SubjectPublicKeyInfo
    )
```

```
    return private_pem, public_pem

# Encrypt message
def encrypt_message(public_key, message):
    encrypted = public_key.encrypt(
        message.encode(),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return encrypted

# Decrypt message
def decrypt_message(private_key, encrypted_message):
    decrypted = private_key.decrypt(
        encrypted_message,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted.decode()

# Example usage
private_key, public_key = generate_rsa_keys()
private_pem, public_pem = save_keys_to_pem(private_key, public_key)

print("Private Key (PEM):", private_pem.decode())
print("Public Key (PEM):", public_pem.decode())

message = "Hello, secure world!"
encrypted_message = encrypt_message(public_key, message)
print("Encrypted Message:", encrypted_message)

decrypted_message = decrypt_message(private_key, encrypted_message)
print("Decrypted Message:", decrypted_message)
```

Blockchain

File Edit View Insert Runtime Tools Help

Commands + Code + Text Connect

```
! pip install cryptography
import cryptography
print("cryptography library is successfully installed!")

from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.serialization import load_pem_private_key, load_pem_public_key
from cryptography.hazmat.primitives.serialization import Encoding, PublicFormat, PrivateFormat, NoEncryption

# Generate RSA key pair
def generate_rsa_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    public_key = private_key.public_key()
    return private_key, public_key

# Save keys to PEM format
def save_keys_to_pem(private_key, public_key):
    private_pem = private_key.private_bytes(
        encoding=Encoding.PEM,
        format=PrivateFormat.PKCS8,
        encryption_algorithm=NoEncryption()
    )
```

Blockchain Practical.ipynb - Colab

```
# Save keys to PEM format
def save_keys_to_pem(private_key, public_key):
    private_pem = private_key.private_bytes(
        encoding=Encoding.PEM,
        format=PrivateFormat.PKCS8,
        encryption_algorithm=NoEncryption()
    )
    public_pem = public_key.public_bytes(
        encoding=Encoding.PEM,
        format=PublicFormat.SubjectPublicKeyInfo
    )
    return private_pem, public_pem

# Encrypt message
def encrypt_message(public_key, message):
    encrypted = public_key.encrypt(
        message.encode(),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return encrypted

# Decrypt message
def decrypt_message(private_key, encrypted_message):
    decrypted = private_key.decrypt(
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Blockchain Practical.ipynb
- File Path:** colab.research.google.com/drive/1cSSpgEm4VV3Nl4ls1WSEe6e8S1GErls#scrollTo=ZoAIVPmf6Nl
- Code Content:**

```
# Decrypt message
def decrypt_message(private_key, encrypted_message):
    decrypted = private_key.decrypt(
        encrypted_message,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted.decode()

# Example usage
private_key, public_key = generate_rsa_keys()
private_pem, public_pem = save_keys_to_pem(private_key, public_key)

print("Private Key (PEM):", private_pem.decode())
print("Public Key (PEM):", public_pem.decode())

message = "Hello, secure world!"
encrypted_message = encrypt_message(public_key, message)
print("Encrypted Message:", encrypted_message)

decrypted_message = decrypt_message(private_key, encrypted_message)
print("Decrypted Message:", decrypted_message)
```
- Toolbar:** Includes icons for back, forward, search, and file operations.
- Header:** Shows the notebook name, file menu (File, Edit, View, Insert, Runtime, Tools, Help), a share button, and a Gemini connection status.
- Bottom Bar:** Shows the Windows taskbar with various pinned applications like File Explorer, Microsoft Edge, and File History.

Output:

PRACTICAL 1 B.

Aim: Allow users to create multiple transactions and display them in an organised format.

Code:

```

from tabulate import tabulate

# Function to add a transaction
def add_transaction(transactions, transaction_id, description, amount):
    transactions.append({"ID": transaction_id, "Description": description,
"Amount": amount})

# Display transactions in a tabular format
def display_transactions(transactions):
    headers = ["Transaction ID", "Description", "Amount"]
    table = [[t["ID"], t["Description"], t["Amount"]] for t in
transactions]
    print(tabulate(table, headers, tablefmt="grid"))

# Main program
def main():
    transactions = [] # List to store transactions
    print("Welcome to the Secure Messaging Transaction System!")

    while True:
        print("\nOptions: ")
        print("1. Add Transaction")
        print("2. Display Transactions")
        print("3. Exit")

        choice = input("\nEnter your choice (1/2/3): ")

        if choice == "1":
            transaction_id = input("Enter Transaction ID: ")
            description = input("Enter Description: ")
            amount = float(input("Enter Amount: "))
            add_transaction(transactions, transaction_id, description,
amount)
            print("Transaction added successfully!")
        elif choice == "2":
            if transactions:
                display_transactions(transactions)
            else:

```

```

        print("No transactions to display!")
    elif choice == "3":
        print("Exiting... Goodbye!")
        break
    else:
        print("Invalid choice. Please select again.")

# Run the program
if __name__ == "__main__":
    main()

```

```

from tabulate import tabulate

# Function to add a transaction
def add_transaction(transactions, transaction_id, description, amount):
    transactions.append({"ID": transaction_id, "Description": description, "Amount": amount})

# Display transactions in a tabular format
def display_transactions(transactions):
    headers = ["Transaction ID", "Description", "Amount"]
    table = [t["ID"], t["Description"], t["Amount"] for t in transactions]
    print(tabulate(table, headers, tablefmt="grid"))

# Main program
def main():
    transactions = [] # List to store transactions
    print("Welcome to the Secure Messaging Transaction System!")

    while True:
        print("\nOptions: ")
        print("1. Add Transaction")
        print("2. Display Transactions")
        print("3. Exit")

        choice = input("\nEnter your choice (1/2/3): ")

        if choice == "1":
            transaction_id = input("Enter Transaction ID: ")
            description = input("Enter Description: ")
            amount = float(input("Enter Amount: "))
            add_transaction(transactions, transaction_id, description, amount)
            print("Transaction added successfully!")

        elif choice == "2":
            if transactions:
                display_transactions(transactions)
            else:
                print("No transactions to display!")

        elif choice == "3":
            print("Exiting... Goodbye!")
            break

    print("Invalid choice. Please select again.")

# Run the program
if __name__ == "__main__":
    main()

```

```

Welcome to the Secure Messaging Transaction System!

Options:
1. Add Transaction
2. Display Transactions
3. Exit

transaction_id = input("Enter Transaction ID: ")
description = input("Enter Description: ")
amount = float(input("Enter Amount: "))
add_transaction(transactions, transaction_id, description, amount)
print("Transaction added successfully!")

# Transaction added successfully!

```

Output:

```
Welcome to the Secure Messaging Transaction System!
Options:
1. Add Transaction
2. Display Transactions
3. Exit

{x}
Enter your choice (1/2/3): 1
Enter Transaction ID: 1200
Enter Description: 1500
Enter Amount: 50000
Transaction added successfully!

Options:
1. Add Transaction
2. Display Transactions
3. Exit

Enter your choice (1/2/3): 2
+-----+-----+
| Transaction ID | Description | Amount |
+=====+=====+=====+
| 1200 | 1500 | 50000 |
+-----+-----+

Options:
1. Add Transaction
2. Display Transactions
3. Exit

Enter your choice (1/2/3): 3
Exiting... Goodbye!
```

PRACTICAL 1 C.

Aim: Create a Python class named Transaction with attributes for sender, receiver, and amount. Implement a method within the class to transfer money from the sender's account to the receiver's account.

Code:

```

class Transaction:
    def __init__(self, sender, receiver, amount):
        self.sender = sender
        self.receiver = receiver
        self.amount = amount
    def transfer(self, accounts):
        """
            Transfers money from the sender's account to the receiver's
            account.
            :param accounts: Dictionary holding account balances for all
            users.
        """
        if self.sender not in accounts:
            print(f"Error: Sender '{self.sender}' does not exist.")
            return
        if self.receiver not in accounts:
            print(f"Error: Receiver '{self.receiver}' does not exist.")
            return
        if accounts[self.sender] < self.amount:
            print(f"Error: Insufficient funds in sender '{self.sender}'"
account.")
            return
        # Perform the transfer
        accounts[self.sender] -= self.amount
        accounts[self.receiver] += self.amount
        print(f"Transfer successful: {self.amount} transferred from
{self.sender} to {self.receiver}.")
# Example usage
if __name__ == "__main__":
    # Sample account balances
    accounts = {
        "Alice": 5000,
        "Bob": 3000,
        "Charlie": 7000,
    }
    # Display initial account balances
    print("Initial account balances:")

```

```

for user, balance in accounts.items():
    print(f"{user}: {balance}")

# Create and perform a transaction
transaction = Transaction("Alice", "Bob", 1500)
transaction.transfer(accounts)

# Display updated account balances
print("\nUpdated account balances:")
for user, balance in accounts.items():
    print(f"{user}: {balance}")

```

The screenshot shows a Jupyter Notebook interface with the following code:

```

class Transaction:
    def __init__(self, sender, receiver, amount):
        self.sender = sender
        self.receiver = receiver
        self.amount = amount

    def transfer(self, accounts):
        """
        Transfers money from the sender's account to the receiver's account.
        :param accounts: Dictionary holding account balances for all users.
        """
        if self.sender not in accounts:
            print(f"Error: Sender '{self.sender}' does not exist.")
            return
        if self.receiver not in accounts:
            print(f"Error: Receiver '{self.receiver}' does not exist.")
            return
        if accounts[self.sender] < self.amount:
            print(f"Error: Insufficient funds in sender '{self.sender}' account.")
            return

        # Perform the transfer
        accounts[self.sender] -= self.amount
        accounts[self.receiver] += self.amount
        print(f"Transfer successful: {self.amount} transferred from {self.sender} to {self.receiver}.")

```

Example usage

The screenshot shows the execution of the script and its output:

```

# Example usage
if __name__ == "__main__":
    # Sample account balances
    accounts = {
        "Alice": 5000,
        "Bob": 3000,
        "Charlie": 7000,
    }

    # Display initial account balances
    print("Initial account balances:")
    for user, balance in accounts.items():
        print(f"{user}: {balance}")

    # Create and perform a transaction
    transaction = Transaction("Alice", "Bob", 1500)
    transaction.transfer(accounts)

    # Display updated account balances
    print("\nUpdated account balances:")
    for user, balance in accounts.items():
        print(f"{user}: {balance}")

```

Initial account balances:

- Alice: 5000
- Bob: 3000
- Charlie: 7000

Updated account balances:

- Alice: 5000
- Bob: 3015
- Charlie: 7000

Output:

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains Python code for a blockchain transaction. The output cell displays the initial account balances, a transaction from Alice to Bob, and the updated account balances after the transfer.

```
"Charlie": 7000,
}

# Display initial account balances
print("Initial account balances:")
for user, balance in accounts.items():
    print(f"{user}: {balance}")

# Create and perform a transaction
transaction = Transaction("Alice", "Bob", 1500)
transaction.transfer(accounts)

# Display updated account balances
print("\nUpdated account balances:")
for user, balance in accounts.items():
    print(f"{user}: {balance}")

Initial account balances:
Alice: 5000
Bob: 3000
Charlie: 7000
Transfer successful: 1500 transferred from Alice to Bob.

Updated account balances:
Alice: 3500
Bob: 4500
Charlie: 7000
```

PRACTICAL 1 D.

Aim: Implement a function to add new blocks to the miner and dump the Blockchain.

Code:

```

import hashlib
import time
class Block:
    def __init__(self, index, previous_hash, timestamp, data, nonce=0):
        self.index = index
        self.previous_hash = previous_hash
        self.timestamp = timestamp
        self.data = data
        self.nonce = nonce
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        """
        Generate a hash for the block using SHA-256.
        """
        block_contents =
f"{self.index}{self.previous_hash}{self.timestamp}{self.data}{self.nonce}"
        return hashlib.sha256(block_contents.encode()).hexdigest()

    def mine_block(self, difficulty):
        """
        Implements proof-of-work by mining a block to match the required
        difficulty.
        """
        target = '0' * difficulty
        while not self.hash.startswith(target):
            self.nonce += 1
            self.hash = self.calculate_hash()
class Blockchain:
    def __init__(self, difficulty=4):
        self.chain = [self.create_genesis_block()]
        self.difficulty = difficulty

    def create_genesis_block(self):
        """
        Create the first block of the blockchain.
        """
        return Block(0, "0", time.time(), "Genesis Block")

```

```
def get_latest_block(self):
    """
    Fetch the latest block in the chain.
    """
    return self.chain[-1]

def add_block(self, data):
    """
    Add a new block to the blockchain.
    """
    latest_block = self.get_latest_block()
    new_block = Block(
        index=latest_block.index + 1,
        previous_hash=latest_block.hash,
        timestamp=time.time(),
        data=data
    )
    new_block.mine_block(self.difficulty)
    self.chain.append(new_block)

def dump_blockchain(self):
    """
    Display all blocks in the blockchain.
    """
    for block in self.chain:
        print(f"Index: {block.index}")
        print(f"Previous Hash: {block.previous_hash}")
        print(f"Timestamp: {block.timestamp}")
        print(f"Data: {block.data}")
        print(f"Nonce: {block.nonce}")
        print(f"Hash: {block.hash}")
        print("-" * 50)

# Example usage
if __name__ == "__main__":
    # Create a blockchain instance
    my_blockchain = Blockchain()

    # Add new blocks
    my_blockchain.add_block("First transaction: Alice pays Bob 50 coins.")
    my_blockchain.add_block("Second transaction: Bob pays Charlie 20
coins.")
    # Dump the blockchain
    print("Blockchain contents:")
    my_blockchain.dump_blockchain()
```

Blockchain

```
import hashlib
import time

class Block:
    def __init__(self, index, previous_hash, timestamp, data, nonce=0):
        self.index = index
        self.previous_hash = previous_hash
        self.timestamp = timestamp
        self.data = data
        self.nonce = nonce
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        """
        Generate a hash for the block using SHA-256.
        """
        block_contents = f'{self.index}{self.previous_hash}{self.timestamp}{self.data}{self.nonce}'
        return hashlib.sha256(block_contents.encode()).hexdigest()

    def mine_block(self, difficulty):
        """
        Implements proof-of-work by mining a block to match the required difficulty.
        """
        target = '0' * difficulty
        while not self.hash.startswith(target):
            self.nonce += 1
            self.hash = self.calculate_hash()

Block
```

```
target = '0' * difficulty
while not self.hash.startswith(target):
    self.nonce += 1
    self.hash = self.calculate_hash()

class Blockchain:
    def __init__(self, difficulty=4):
        self.chain = [self.create_genesis_block()]
        self.difficulty = difficulty

    def create_genesis_block(self):
        """
        Create the first block of the blockchain.
        """
        return Block(0, "0", time.time(), "Genesis Block")

    def get_latest_block(self):
        """
        Fetch the latest block in the chain.
        """
        return self.chain[-1]

    def add_block(self, data):
        """
        Add a new block to the blockchain.
        """
        latest_block = self.get_latest_block()
        new_block = Block(
            latest_block.index + 1,
            latest_block.hash,
            time.time(),
            data,
            latest_block.nonce
        )
        new_block.mine_block(self.difficulty)
        self.chain.append(new_block)

Blockchain
```

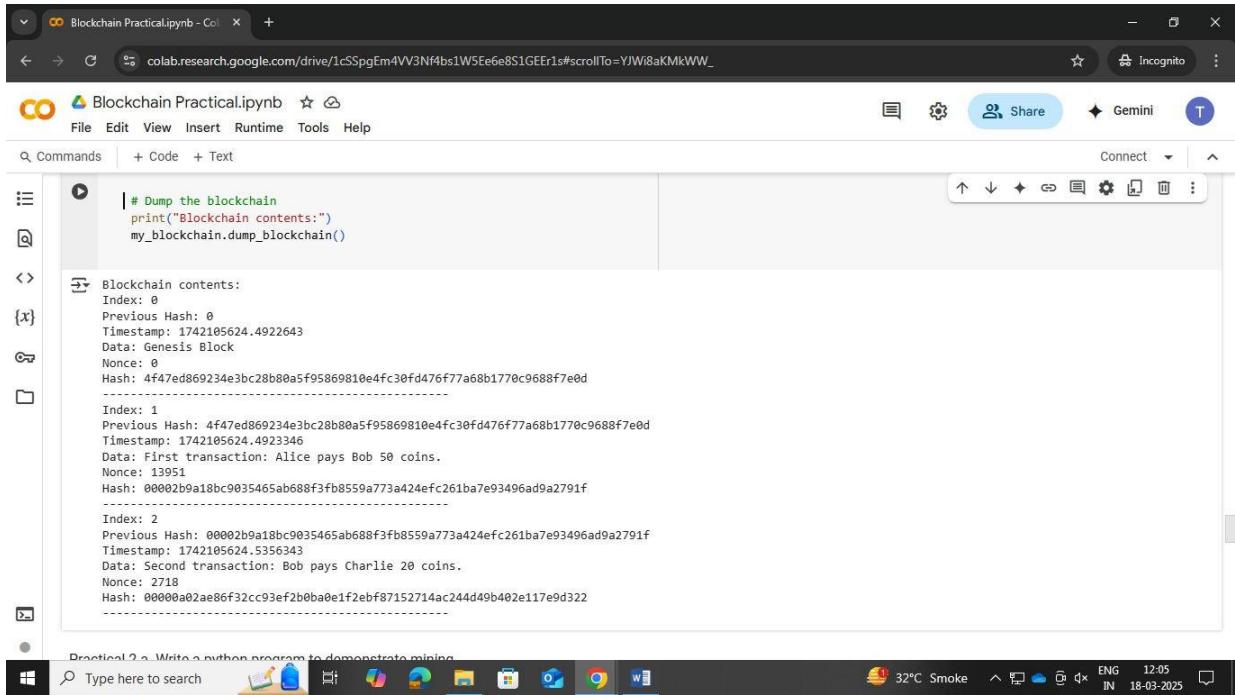
```
latest_block = self.get_latest_block()
new_block = Block(
    latest_block.index + 1,
    latest_block.hash,
    time.time(),
    data,
    latest_block.nonce
)
new_block.mine_block(self.difficulty)
self.chain.append(new_block)

def dump_blockchain(self):
    """
    Display all blocks in the blockchain.
    """
    for block in self.chain:
        print(f"Index: {block.index}")
        print(f"Previous hash: {block.previous_hash}")
        print(f"Timestamp: {block.timestamp}")
        print(f"Data: {block.data}")
        print(f"Nonce: {block.nonce}")
        print(f"Hash: {block.hash}")
        print("-" * 50)

# Example usage
if __name__ == "__main__":
    # Create a blockchain instance
    my_blockchain = Blockchain()

    # Add new blocks
    my_blockchain.add_block("First transaction: Alice pays Bob 50 coins.")
    my_blockchain.add_block("Second transaction: Bob pays Charlie 20 coins.")

    # Dump the blockchain
    print("Blockchain contents:")
    my_blockchain.dump_blockchain()
```

Output:

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains the following Python code:

```
# Dump the blockchain
print("Blockchain contents:")
my_blockchain.dump_blockchain()
```

The output cell displays the blockchain contents, which include three blocks. The first block is the Genesis Block, containing a single transaction from Alice to Bob. The second block contains a transaction from Bob to Charlie. The third block is empty.

```
Blockchain contents:
Index: 0
Previous Hash: 0
Timestamp: 1742105624.4922643
Data: Genesis Block
Nonce: 0
Hash: 4f47ed869234e3bc28b80a5f95869810e4fc30fd476f77a68b1770c9688f7e0d
-----
Index: 1
Previous Hash: 4f47ed869234e3bc28b80a5f95869810e4fc30fd476f77a68b1770c9688f7e0d
Timestamp: 1742105624.4923346
Data: First transaction: Alice pays Bob 50 coins.
Nonce: 13951
Hash: 00002b9a18bc9035465ab688f3fb8559a773a424efc261ba7e93496ad9a2791f
-----
Index: 2
Previous Hash: 00002b9a18bc9035465ab688f3fb8559a773a424efc261ba7e93496ad9a2791f
Timestamp: 1742105624.5356343
Data: Second transaction: Bob pays Charlie 20 coins.
Nonce: 2718
Hash: 00000a02ae86f32cc93ef2b0ba0e1f2ebf87152714ac244d49b402e117e9d322
```

PRACTICAL 2 A.

Aim: Write a python program to demonstrate mining.

Code:

```

import hashlib
import time

def mine(block_data, difficulty):
    prefix = '0' * difficulty    # Define the difficulty level (number of
leading zeros)
    nonce = 0    # Nonce starts at 0

    print("Mining in progress...")
    start_time = time.time()

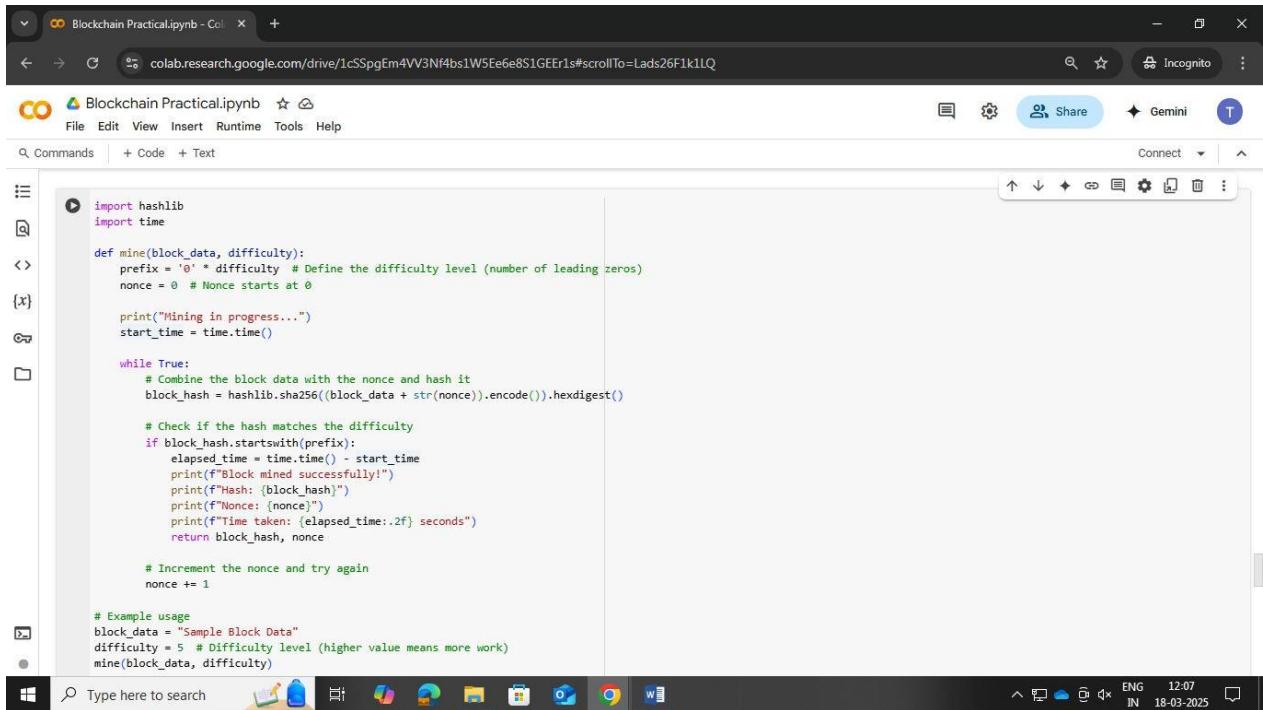
    while True:
        # Combine the block data with the nonce and hash it
        block_hash = hashlib.sha256((block_data +
str(nonce)).encode()).hexdigest()

        # Check if the hash matches the difficulty
        if block_hash.startswith(prefix):
            elapsed_time = time.time() - start_time
            print(f"Block mined successfully!")
            print(f"Hash: {block_hash}")
            print(f"Nonce: {nonce}")
            print(f"Time taken: {elapsed_time:.2f} seconds")
            return block_hash, nonce

        # Increment the nonce and try again
        nonce += 1

# Example usage
block_data = "Sample Block Data"
difficulty = 5    # Difficulty level (higher value means more work)
mine(block_data, difficulty)

```



The screenshot shows a Google Colab notebook titled "Blockchain Practical.ipynb". The code implements a simple blockchain mining algorithm using the hashlib library. It defines a function `mine` that takes block data and difficulty as inputs. The function loops until it finds a hash starting with a specified number of zeros (the prefix). It prints the hash, nonce, and time taken. An example usage is shown at the bottom.

```

import hashlib
import time

def mine(block_data, difficulty):
    prefix = '0' * difficulty # Define the difficulty level (number of leading zeros)
    nonce = 0 # Nonce starts at 0

    print("Mining in progress...")
    start_time = time.time()

    while True:
        # Combine the block data with the nonce and hash it
        block_hash = hashlib.sha256((block_data + str(nonce)).encode()).hexdigest()

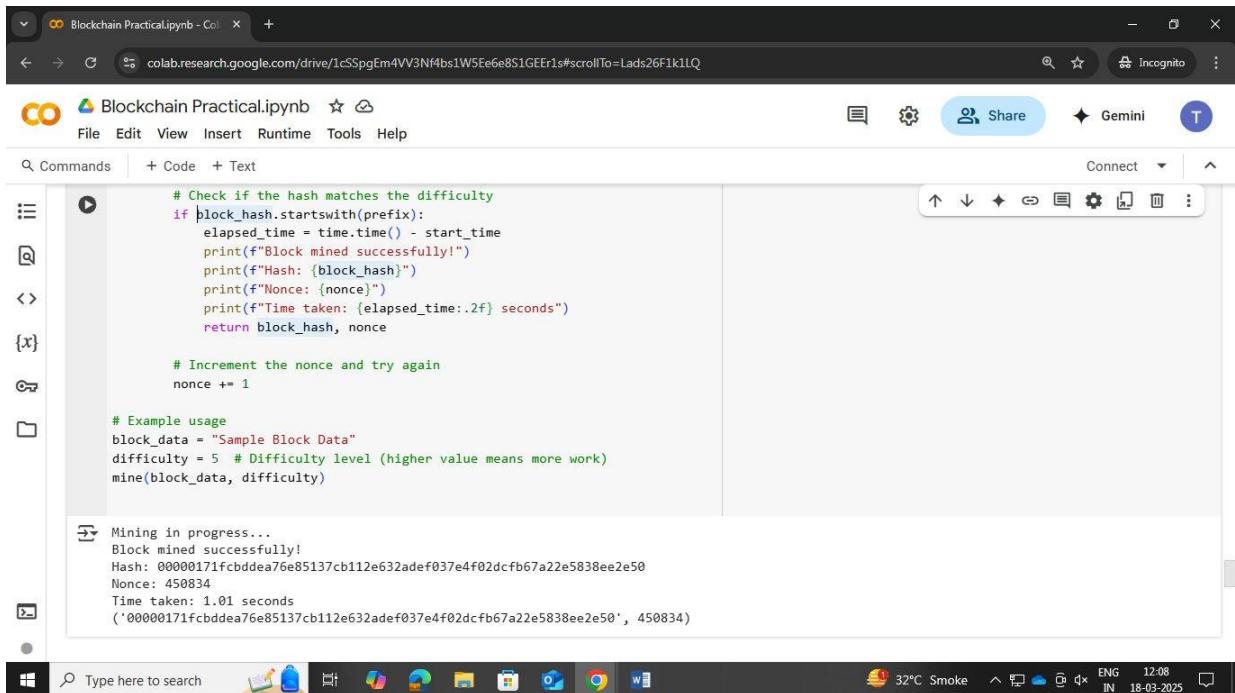
        # Check if the hash matches the difficulty
        if block_hash.startswith(prefix):
            elapsed_time = time.time() - start_time
            print(f"Block mined successfully!")
            print(f"Hash: {block_hash}")
            print(f"Nonce: {nonce}")
            print(f"Time taken: {elapsed_time:.2f} seconds")
            return block_hash, nonce

        # Increment the nonce and try again
        nonce += 1

# Example usage
block_data = "Sample Block Data"
difficulty = 5 # Difficulty level (higher value means more work)
mine(block_data, difficulty)

```

Output:



The screenshot shows the output of the mining process in Google Colab. The terminal window displays the mining progress, successful hash generation, and the time taken. The output is identical to the one shown in the previous screenshot.

```

# Check if the hash matches the difficulty
if block_hash.startswith(prefix):
    elapsed_time = time.time() - start_time
    print(f"Block mined successfully!")
    print(f"Hash: {block_hash}")
    print(f"Nonce: {nonce}")
    print(f"Time taken: {elapsed_time:.2f} seconds")
    return block_hash, nonce

# Increment the nonce and try again
nonce += 1

# Example usage
block_data = "Sample Block Data"
difficulty = 5 # Difficulty level (higher value means more work)
mine(block_data, difficulty)

Mining in progress...
Block mined successfully!
Hash: 00000171fcbbdea76e85137cb112e632adef037e4f02dcfb67a22e5838ee2e50
Nonce: 450834
Time taken: 1.01 seconds
('00000171fcbbdea76e85137cb112e632adef037e4f02dcfb67a22e5838ee2e50', 450834)

```

PRACTICAL 2 B.

Aim: Demonstrate the use of the Bitcoin Core API to interact with a Bitcoin Core node.

Code:

```
server=1

rpcuser=your_rpc_user

rpcpassword=your_rpc_password

rpcport=8332

rpcallowip=127.0.0.1

bitcoind –daemon

pip install requests

import requests

import json

from requests.auth import HTTPBasicAuth

# Bitcoin Core RPC settings

rpc_url = "http://127.0.0.1:8332"

rpc_user = "your_rpc_user" # Replace with your RPC username

rpc_password = "your_rpc_password" # Replace with your RPC password

# Define a function to send RPC requests

def bitcoin_rpc(method, params=None):

    headers = {'content-type': 'application/json'}

    # Prepare the RPC request payload

    payload = {

        "jsonrpc": "1.0",

        "id": "curltest",

        "method": method,

        "params": params or []}
```

```
}

# Send the POST request

response = requests.post(rpc_url, data=json.dumps(payload), headers=headers,
auth=HTTPBasicAuth(rpc_user, rpc_password))

if response.status_code == 200:
    return response.json()
else:
    raise Exception(f"Error: {response.status_code} - {response.text}")

# Example 1: Get the current block count

block_count = bitcoin_rpc("getblockcount")

print("Block Count:", block_count['result'])

# Example 2: Get blockchain information

blockchain_info = bitcoin_rpc("getblockchaininfo")

print("Blockchain Info:", blockchain_info['result'])

# Example 3: Get the wallet balance

wallet_balance = bitcoin_rpc("getbalance")

print("Wallet Balance:", wallet_balance['result'])

# Example 4: Get the current network difficulty

difficulty = bitcoin_rpc("getdifficulty")

print("Network Difficulty:", difficulty['result'])
```

```
# Example 5: Send a raw transaction (assuming you have a raw transaction to send)

# raw_tx = "<raw_transaction_data>"

# tx_id = bitcoin_rpc("sendrawtransaction", [raw_tx])

# print("Transaction ID:", tx_id['result'])
```

Output:

Block Count: 768914

Wallet Balance: 0.02346789

Network Difficulty: 218938744019.67

Transaction ID: 1d5f8b927a9d7a64d83a57ac64d0b173d5e7a4932d8365f07c50e85826f1b27b

Error Handling:

If something goes wrong, you'll receive error messages from the Bitcoin Core node.

Invalid RPC call (wrong method name):

```
plaintext
CopyEdit
Error: -32601 - Method not found
```

Incorrect RPC credentials (username/password mismatch):

plaintext
CopyEdit
Error: 401 - Unauthorized

If the node isn't fully synced and you attempt a `getblock` request:

```
plaintext
CopyEdit
Error: -8 - Block not found
```

PRACTICAL 2 C.

Aim: Demonstrating the process of running a blockchain node on your local machine

Code:

Running a blockchain node on your local machine allows you to interact directly with the blockchain network. Here we will go through the process of setting up a **Bitcoin Core node**, which is one of the most widely used full-node implementations of the Bitcoin protocol. This setup will help you run a **Bitcoin full node** locally and interact with the blockchain.

Steps to Run a Bitcoin Node Locally

1. Install Bitcoin Core

Bitcoin Core is the reference implementation of the Bitcoin protocol. To run a full Bitcoin node, you need to install Bitcoin Core on your machine.

Download Bitcoin Core:

- Visit the official website to download Bitcoin Core for your operating system:
[Download Bitcoin Core](#)

Installation:

- Follow the installation instructions for your operating system (Windows, Linux, macOS).

2. Configure Bitcoin Core

Once you have Bitcoin Core installed, you need to configure it to run as a full node and allow RPC (Remote Procedure Call) for interaction.

[Create or Edit the bitcoin.conf file:](#)

- The configuration file is usually located in the Bitcoin data directory:
 - **Linux:** `~/.bitcoin/bitcoin.conf`
 - **macOS:** `~/Library/Application Support/Bitcoin/bitcoin.conf`
 - **Windows:** `C:\Users\YourUsername\AppData\Roaming\Bitcoin\bitcoin.conf`

If the file doesn't exist, create it.

Configure the File:

Open or create the bitcoin.conf file and add the following lines for basic setup:

```
# Bitcoin Core Configuration File
server=1
rpcuser=your_rpc_user
rpcpassword=your_rpc_password
rpcport=8332
rpcallowip=127.0.0.1
listen=1
maxconnections=125
datadir=/path/to/bitcoin/data
txindex=1
```

Explanation of each line:

- server=1: This allows the node to accept RPC commands.
- rpcuser: Username for authentication when sending RPC commands.
- rpcpassword: Password for the above username.
- rpcport: Port for RPC communication (default is 8332).
- rpcallowip=127.0.0.1: Allow RPC connections from the local machine only (for security).
- listen=1: Allows the node to accept incoming connections from other nodes.
- maxconnections=125: Maximum number of connections the node will allow.
- datadir: Directory where blockchain data will be stored (you can customize this).
- txindex=1: Enables transaction index, useful for querying transactions by ID.

3. Start Bitcoin Core

To start Bitcoin Core, you can run the following command depending on your operating system:

- **Linux/macOS:**
- bitcoind -daemon
- **Windows:**
Double-click on bitcoind.exe (if installed via the Windows installer) or run the following in the command prompt:
- bitcoind.exe -daemon

This starts the Bitcoin daemon in the background.

4. Sync the Blockchain

When you first start Bitcoin Core, it will begin to download the entire Bitcoin blockchain. This process is known as **syncing**. This can take **several days** depending on your internet speed and hardware.

The blockchain data is stored in the Bitcoin data directory (`datadir`). The blockchain can grow large (currently around 500 GB), so ensure you have enough storage space available.

You can monitor the sync progress via the Bitcoin Core graphical interface or by using RPC commands.

5. Check Sync Progress

To check the synchronization status of your node, you can use the following RPC command:

- Use a **Python script** or **command line** to interact with Bitcoin Core RPC.

[Example Python Script to Check Block Height:](#)

```
import requests
import json
from requests.auth import HTTPBasicAuth

rpc_url = "http://127.0.0.1:8332"
rpc_user = "your_rpc_user" # Replace with your RPC username
rpc_password = "your_rpc_password" # Replace with your RPC password

def bitcoin_rpc(method, params=None):
    headers = {'content-type': 'application/json'}
    payload = {
        "jsonrpc": "1.0",
        "id": "curltest",
        "method": method,
        "params": params or []
    }
    response = requests.post(rpc_url, data=json.dumps(payload), headers=headers, auth=HTTPBasicAuth(rpc_user, rpc_password))
    return response.json()

# Example: Get block count to check sync status
block_count = bitcoin_rpc("getblockcount")
print("Block Count:", block_count['result'])
```

This Python script queries the `getblockcount` method, which returns the current block height. If the node isn't fully synced, the block count will be lower than the latest block on the network.

6. Interacting with the Bitcoin Network

Once your Bitcoin node is synced, you can start interacting with the Bitcoin blockchain. Here are some common actions you might want to perform:

[Get Blockchain Information:](#)

```
blockchain_info = bitcoin_rpc("getblockchaininfo")
print("Blockchain Info:", blockchain_info['result'])
```

This will return details like the current block height, the best block hash, and other useful information about the blockchain.

Get a New Address:

If you want to create a new Bitcoin address to receive funds, use:

```
new_address = bitcoin_rpc("getnewaddress")
print("New Address:", new_address['result'])
```

This generates a new Bitcoin address from your wallet.

Send Bitcoin:

To send Bitcoin from your wallet to another address, you need to use the `sendtoaddress` method.
Example:

```
tx_id = bitcoin_rpc("sendtoaddress", ["1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa", 0.001])
print("Transaction ID:", tx_id['result'])
```

This sends 0.001 BTC to the address 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa.

7. Monitor the Node

You can monitor the status of your node using Bitcoin Core's graphical user interface (GUI) or via RPC commands. Bitcoin Core also provides detailed logs that can help troubleshoot any issues.

Check Node's Version:

```
version_info = bitcoin_rpc("getnetworkinfo")
print("Version Info:", version_info['result'])
```

This will return information like the current Bitcoin version and network info.

8. Closing Bitcoin Core

Once you're done using the Bitcoin Core node, you can stop it gracefully by calling:

```
bitcoin-cli stop
```

PRACTICAL 2 D.

Aim: Demonstrate mining using geth on your private network.

Code:

Steps to Mine on a Private Ethereum Network using Geth

1. Install Geth

First, you need to install Geth, which is the Go-based Ethereum client.

[Install Geth:](#)

- **Windows:**
Download the latest Geth installer from the [official Geth GitHub release page](#) and follow the installation instructions.
- **macOS:**
If you have `brew` installed, you can install Geth with:
 - `brew tap ethereum/ethereum`
 - `brew install ethereum`
- **Linux:**
On Ubuntu/Debian:
 - `sudo add-apt-repository -y ppa:ethereum/ethereum`
 - `sudo apt-get update`
 - `sudo apt-get install geth`

Once installed, you can check if it's working by running:

```
geth version
```

2. Create a Genesis Block for Your Private Network

To start a private Ethereum network, you need to define the **genesis block** (the first block in the blockchain). This is done by creating a `genesis.json` file.

[Create `genesis.json`:](#)

Create a `genesis.json` file that will define the parameters of your private blockchain. Here's an example of what the `genesis.json` file might look like:

```
{
  "config": {
    "chainId": 1337, // A unique chain ID for the private
network
    "homesteadBlock": 0,
    "daoForkBlock": 0,
```

```

    "eip155Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "muirGlacierBlock": 0,
    "berlinBlock": 0,
    "londonBlock": 0
  },
  "alloc": {},
  "coinbase": "0x000000000000000000000000000000000000000000000000000000000000000",
  "difficulty": "0x20000",
  "gasLimit": "0x8000000"
}

```

Explanation:

- `chainId`: This is a unique identifier for your private network. Ethereum mainnet's chain ID is 1, and test networks have their own chain IDs.
- `alloc`: You can pre-allocate some Ether to specific accounts. In this case, it's left empty.
- `difficulty`: Set the mining difficulty. We can set this to a low value (0x20000) so mining is quick on your private network.
- `gasLimit`: The gas limit for blocks (default is set to 0x8000000).

Initialize the Genesis Block:

Run the following command to initialize the private network using the `genesis.json` file:

```
geth init genesis.json --datadir ./privateChain
```

This initializes the private blockchain and creates the necessary data directory (`./privateChain`).

3. Start Your Ethereum Node

Now that your private network has been initialized, you can start the Geth node.

Run the following command to start your Ethereum node on the private network:

```
geth --networkid 1337 --datadir ./privateChain --mine --miner.threads=1 --
http --http.addr "0.0.0.0" --http.port 8545 --http.api
"personal,eth,web3,miner,net" --allow-insecure-unlock
```

Explanation of the parameters:

- `--networkid 1337`: This ensures you're using your private network (use the chain ID you specified in the genesis file).
- `--datadir ./privateChain`: Points to the data directory for your blockchain.
- `--mine`: This enables mining.

- `--miner.threads=1`: Number of CPU threads to mine on (you can increase this if you have a more powerful machine).
- `--http`: Enables the HTTP RPC server (useful for interacting with your node programmatically).
- `--http.addr "0.0.0.0"`: Allows RPC connections from any machine (or use `"127.0.0.1"` for local-only access).
- `--http.port 8545`: Specifies the HTTP RPC port.
- `--http.api "personal,eth,web3,miner,net"`: This allows the specified APIs to be accessible over HTTP (necessary for interacting with the miner and the blockchain).
- `--allow-insecure-unlock`: This allows you to unlock accounts for mining, but use it carefully because it can expose your accounts.

4. Unlock Your Mining Account

In order to mine on your private network, you'll need an unlocked account. You can either create an account or unlock an existing one.

[Create a New Account:](#)

Run this command to create a new Ethereum account:

```
geth account new --datadir ./privateChain
```

This will generate a new account and give you an address.

[Unlock the Account:](#)

To unlock the account for mining (it's required to mine on your node), run:

```
geth attach ipc:./privateChain/geth.ipc
```

This will open the Geth JavaScript console. Then, unlock the account using the following command:

```
personal.unlockAccount(eth.accounts[0], "your_password", 0)
```

Where `eth.accounts[0]` is the first account in your list and `"your_password"` is the password you set during account creation. The `0` specifies the unlock duration (0 means permanently unlocked until you shut down the node).

5. Start Mining

Now that your account is unlocked and the Geth node is running, mining should begin automatically.

You can check the mining process by viewing the logs in the console, and you should see new blocks being mined at regular intervals (because we set a low difficulty in the `genesis.json` file).

If you're running the node with the `--mine` flag, mining should be ongoing. You should see new blocks being mined and their corresponding block hashes in the logs. The mining reward (in ETH) will be credited to your account.

You can also check the mining status with this RPC call:

```
eth.mining
```

This should return `true` if mining is in progress.

6. Monitor Mining and Network Status

You can use the following commands within the **Geth JavaScript console** to get more information about the current mining status:

- **Check current block number:**
`eth.blockNumber`
- **Check the coinbase (miner's address):**
`eth.coinbase`
- **Check balance of the account:**
`web3.fromWei(eth.getBalance(eth.coinbase), "ether")`

7. Interact with the Ethereum Network

If you want to interact with the private Ethereum network from a web interface, you can use **Web3.js** (JavaScript library) or **Web3.py** (Python library) to make RPC calls to the network.

[Example Web3.js Code:](#)

Here's an example of how to interact with the node using Web3.js:

```
const Web3 = require('web3');

// Connect to the local Ethereum node
const web3 = new Web3('http://localhost:8545');
web3.eth.getBlockNumber()
  .then(console.log); // Prints the latest block number
```

8. Stopping the Node

To stop your Ethereum node, press `Ctrl+C` in the terminal where Geth is running.

```
admin.stopRPC()
```

PRACTICAL 3 A.

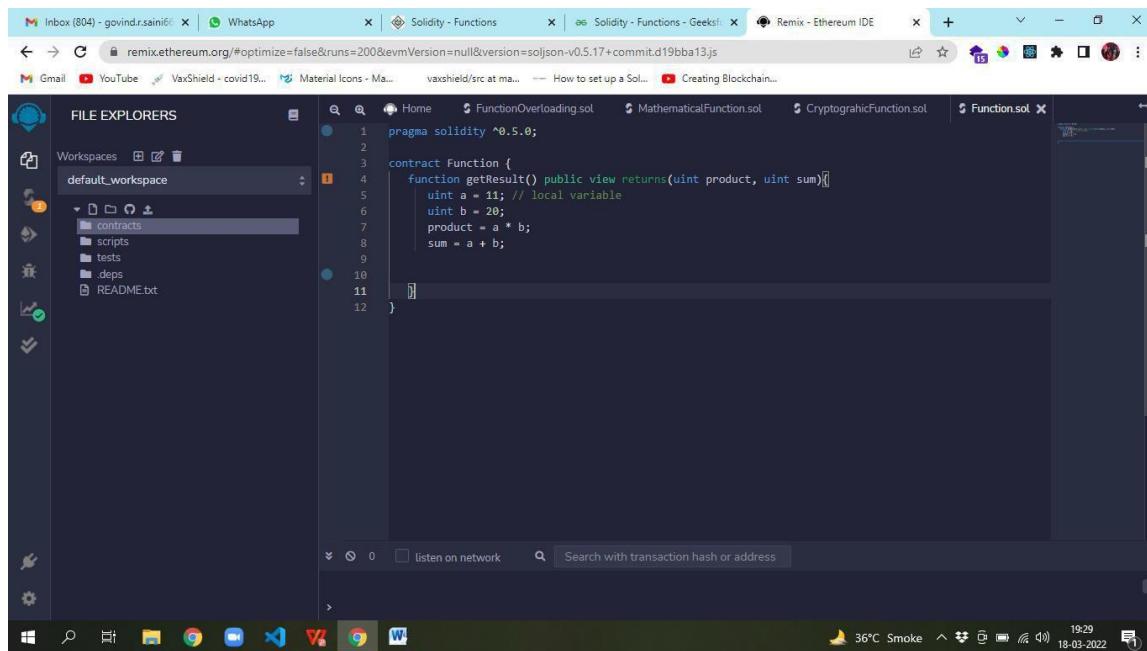
Aim: Write a Solidity program that demonstrates various types of functions including regular functions, view functions, pure functions, and the fallback function.

Code:

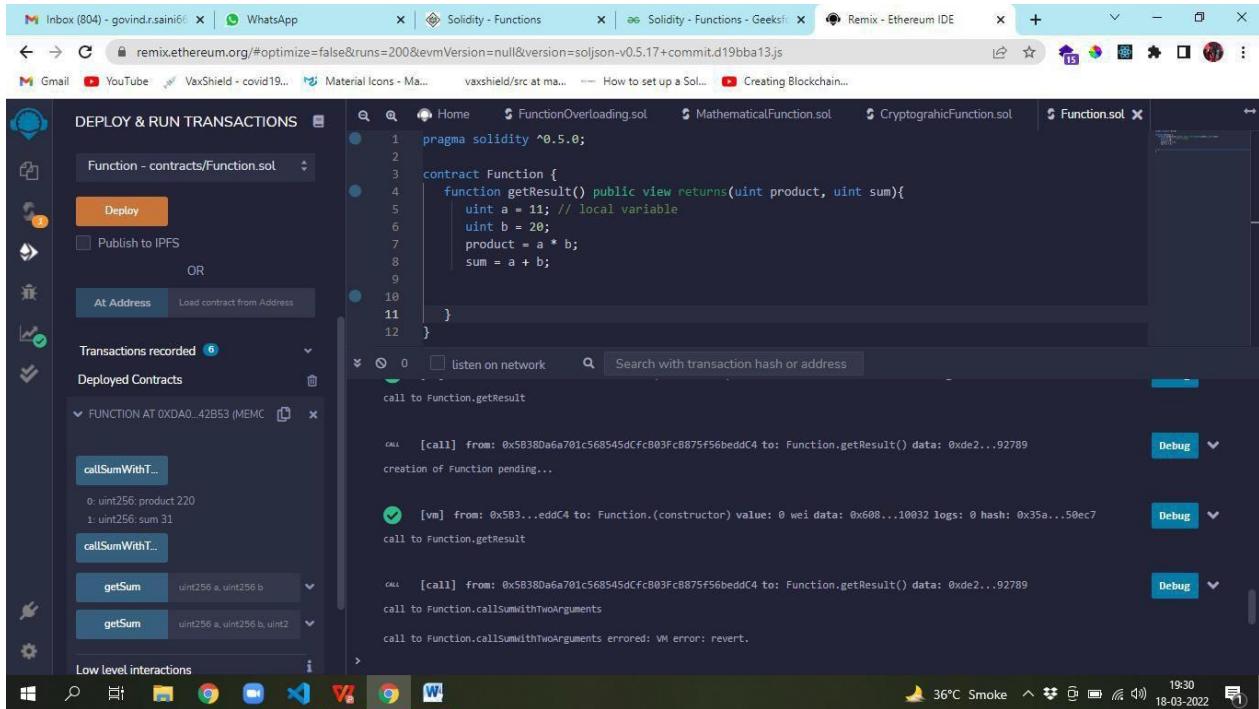
Regular Function

```
pragma solidity ^0.5.0;

contract Function {
    function getResult() public view returns (uint product, uint sum){
        uint a = 11;
        uint b = 20;
        product = a * b;
        sum = a + b;
    }
}
```



Output:



View Function

```
pragma solidity ^0.5.0;
```

```
contract ViewFunction {
```

```
    uint num1 = 2;
```

```
    uint num2 = 4;
```

```
function getResult()
```

```
public view returns(
```

```
    uint product, uint sum) {
```

```
    uint num1 = 10;
```

```
    uint num2 = 16;
```

```
    product = num1 * num2;
```

```

sum = num1 + num2;
}

}

```

```

pragma solidity ^0.5.0;

contract ViewFunction {
    uint num1 = 2;
    uint num2 = 4;

    function getResult()
        public view returns(
            uint product, uint sum){
            uint num1 = 10;
            uint num2 = 16;
            product = num1 * num2;
            sum = num1 + num2;
    }
}

```

Output:

DEPLOY & RUN TRANSACTIONS

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 6

Deployed Contracts

VIEWFUNCTION AT 0X7EF...8CB47 (ViewFunction)

VIEWFUNCTION AT 0XA0...42B53 (ViewFunction)

creation of ViewFunction pending...

getResult

0: uint256: product 160
1: uint256: sum 26

Low level interactions

TRANSACTION

[call] from: 0x5B380a6a701c568545dCfcB03FcB875f56beddC4 to: ViewFunction.getResult() data: 0xde2...92789

Pure Function

```
pragma solidity ^0.5.0;

contract PureFunction {

function getResult()

public pure returns (

uint product, uint sum) {

uint num1 = 2;

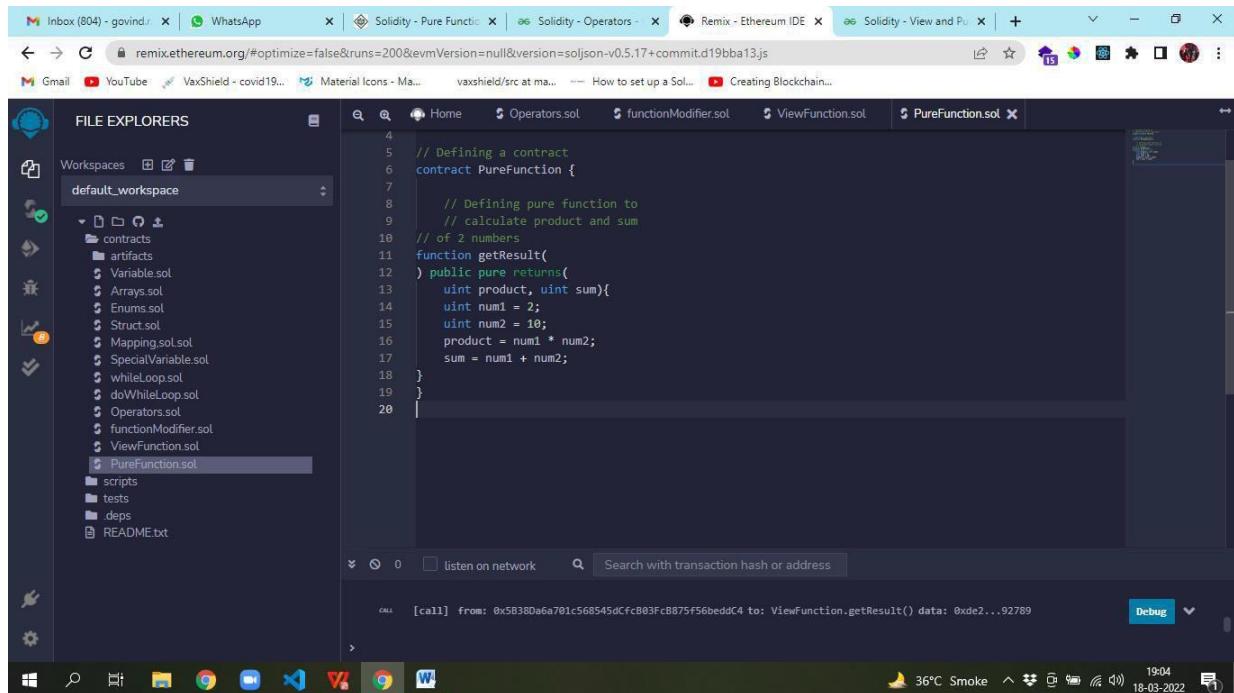
uint num2 = 10;

product = num1 * num2;

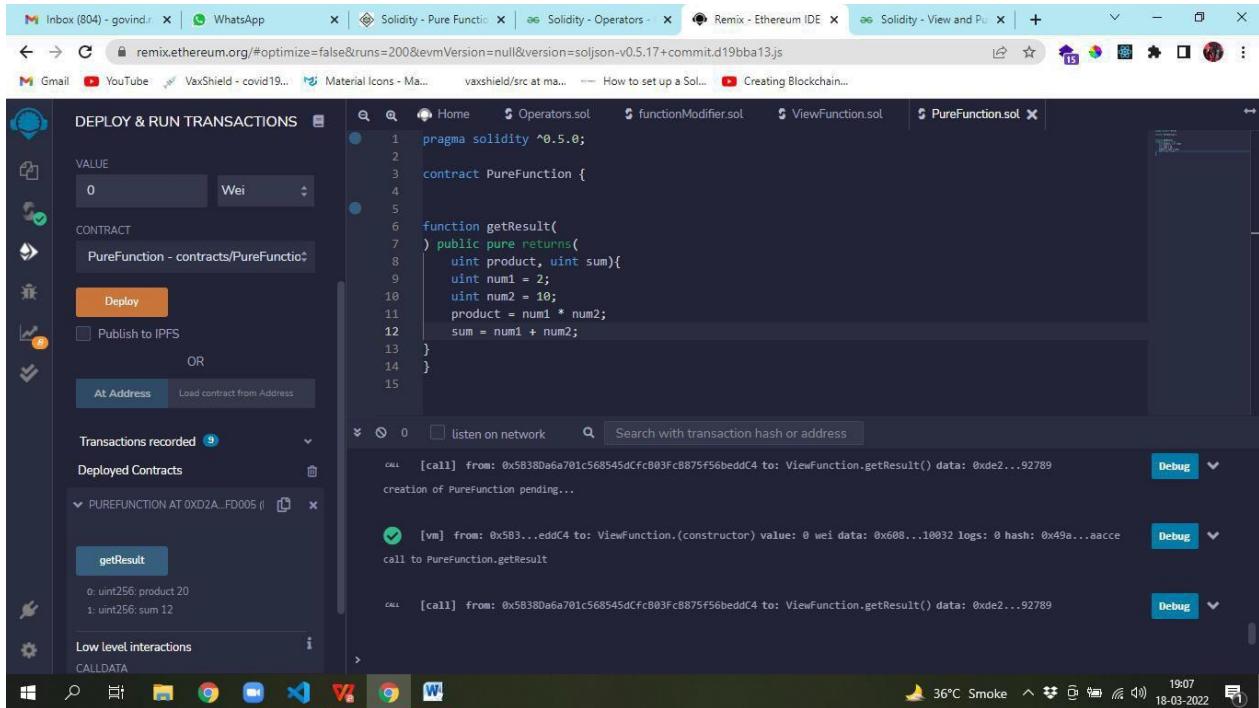
sum = num1 + num2;

}

}
```



Output:



Fallback Function

```
pragma solidity ^0.5.0;

contract FallbackFunction {
    uint public x ;

    function() external { x = 1; }

}

contract Sink {
    function() external payable { }
}

contract Caller {
    function callTest (Test test) public returns (bool) {
        (bool success, ) =
            address(test).call(abi.encodeWithSignature("nonExistingFunction()"));
    }
}
```

```

require(success);

address payable testPayable = address(uint160(address(test)));

return (testPayable.send(2 ether));

}

function callSink (Sink sink) public returns (bool) {

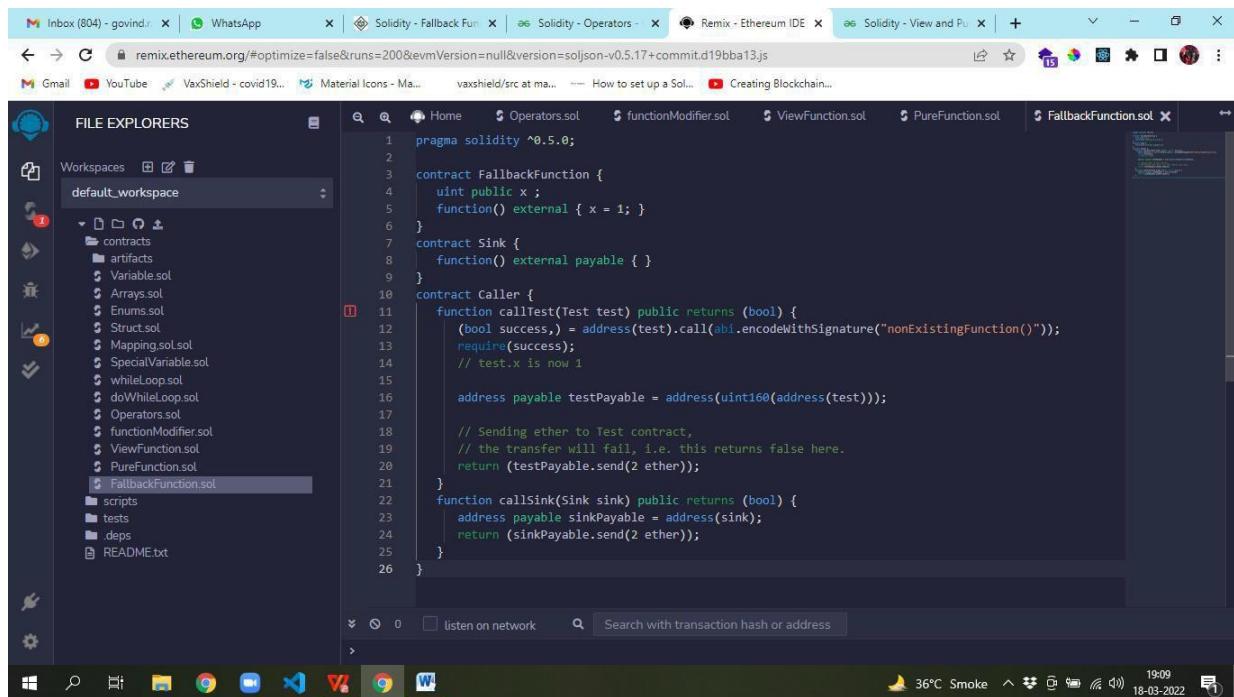
address payable sinkPayable = address(sink);

return (sinkPayable.send(2 ether));

}

}

```



The screenshot shows the Remix Ethereum IDE interface. The top navigation bar includes tabs for Remix - Ethereum IDE and Solidity - View and Run. The main workspace displays the Solidity code for `FallbackFunction.sol`. The code defines three contracts: `FallbackFunction`, `Sink`, and `Caller`. The `Caller` contract contains two functions: `callTest` and `callSink`. The `callTest` function calls the non-existing function `x` on the `testPayable` contract and sends 2 ether to it. The `callSink` function calls the `Sink` contract and sends 2 ether to it. The left sidebar shows the file structure under `default_workspace`, including `contracts`, `artifacts`, and several other Solidity files like `Variable.sol`, `Arrays.sol`, etc.

```

1 pragma solidity ^0.5.0;
2
3 contract FallbackFunction {
4     uint public x ;
5     function() external { x = 1; }
6 }
7 contract Sink {
8     function() external payable { }
9 }
10 contract Caller {
11     function callTest(Test test) public returns (bool) {
12         (bool success,) = address(test).call(abi.encodeWithSignature("nonExistingFunction()"));
13         require(success);
14         // test.x is now 1
15
16         address payable testPayable = address(uint160(address(test)));
17
18         // Sending ether to Test contract,
19         // the transfer will fail, i.e. this returns false here.
20         return (testPayable.send(2 ether));
21     }
22     function callSink(Sink sink) public returns (bool) {
23         address payable sinkPayable = address(sink);
24         return (sinkPayable.send(2 ether));
25     }
26 }

```

Output:

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is open, showing the deployment of a 'Caller' contract that interacts with a 'FallbackFunction' contract. The main workspace displays the Solidity code for these contracts. The 'FallbackFunction.sol' tab is active, containing the following code:

```
pragma solidity ^0.5.0;

contract FallbackFunction {
    uint public x;
    function() external { x = 1; }
}

contract Sink {
    function() external payable {}
}

contract Caller {
    function callTest(FallbackFunction test) public returns (bool) {
        (bool success,) = address(test).call(abi.encodeWithSignature("nonExistingFunction()"));
        require(success);
        // test.x is now 1
    }

    address payable testPayable = address(uint160(address(test)));
}
```

The 'Transactions recorded' section shows a recent transaction from the 'Caller' contract to the 'testPayable' address. The status bar at the bottom indicates a system temperature of 36°C and a date of 18-03-2022.

PRACTICAL 3 B.

Aim: Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.

Code:

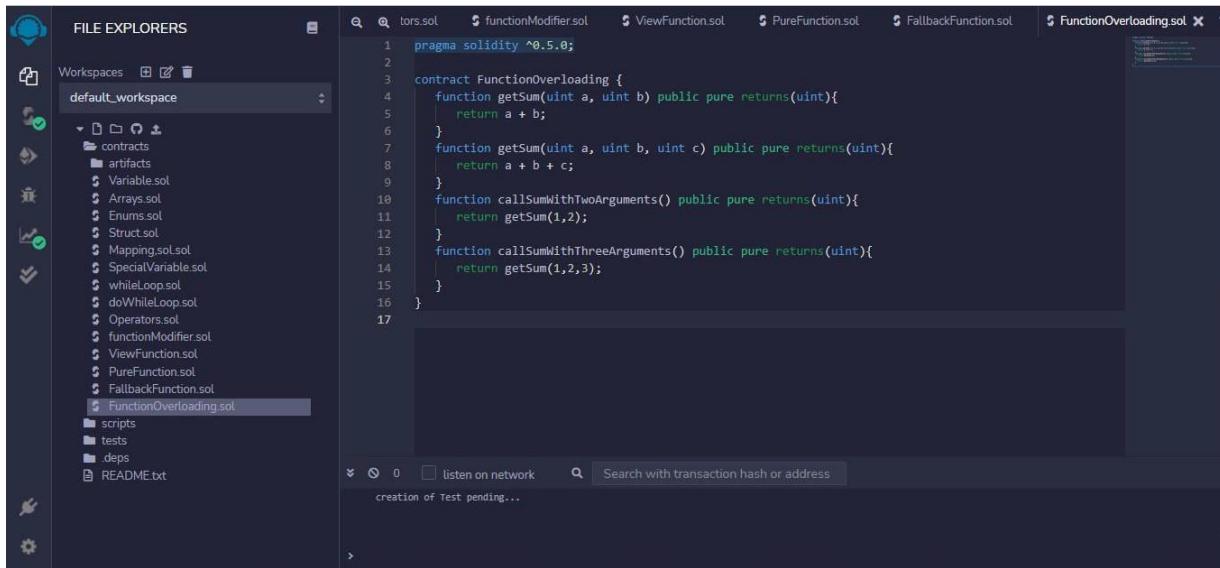
// Function overloading

```
pragma solidity ^0.5.0;

contract DemoContract {

    function calculate(uint a, uint b) public pure returns (uint) {
        return a + b;
    }

    function calculate(uint a, uint b, uint c) public pure returns (uint) {
        return a + b + c;
    }
}
```



The screenshot shows the Solidity IDE interface. On the left, the FILE EXPLORERS panel displays a workspace named 'default_workspace' containing several Solidity files: contracts, artifacts, Variable.sol, Arrays.sol, Enums.sol, Struct.sol, Mapping.sol.sol, SpecialVariable.sol, whileLoop.sol, doWhileLoop.sol, Operators.sol, functionModifier.sol, ViewFunction.sol, PureFunction.sol, FallbackFunction.sol, and FunctionOverloading.sol. The 'FunctionOverloading.sol' file is currently selected and shown in the main editor area. The code in the editor is as follows:

```
1 pragma solidity ^0.5.0;
2
3 contract FunctionOverloading {
4     function getSum(uint a, uint b) public pure returns(uint){
5         return a + b;
6     }
7     function getSum(uint a, uint b, uint c) public pure returns(uint){
8         return a + b + c;
9     }
10    function callSumWithTwoArguments() public pure returns(uint){
11        return getSum(1,2);
12    }
13    function callSumWithThreeArguments() public pure returns(uint){
14        return getSum(1,2,3);
15    }
16 }
17
```

The status bar at the bottom indicates 'creation of Test pending...'.

Output

```

1 pragma solidity ^0.5.0;
2
3 contract FunctionOverloading {
4     function getSum(uint a, uint b) public pure returns(uint){
5         return a + b;
6     }
7     function getSum(uint a, uint b, uint c) public pure returns(uint){
8         return a + b + c;
9     }
10    function callSumWithTwoArguments() public pure returns(uint){
11        return getSum(1,2);
12    }
13    function callSumWithThreeArguments() public pure returns(uint){
14        return getSum(1,2,3);
15    }
}

```

Type the library name to see available commands.
creation of FunctionOverloading pending...

[call] from: 0x5E9B0a6a701c568545dFcB03FcB8875f56beddC4 to: FunctionOverloading.callSumWithThreeArguments()
data: 0xd20...7419b
call to FunctionOverloading.callSumWithThreeArguments

[call] from: 0x5E9B0a6a701c568545dFcB03FcB8875f56beddC4 to: FunctionOverloading.callSumWithTwoArguments()
data: 0xd20...7419b
call to FunctionOverloading.callSumWithTwoArguments

// Mathematical functions

```

function getSquareRoot(uint x) public pure returns (uint) {

    return uint(sqrt(x));
}

function power(uint base, uint exponent) public pure returns (uint) {

    return base ** exponent;
}

```

```

1 pragma solidity ^0.5.0;
2
3 contract MathematicalFunction {
4     function callAddMod() public pure returns(uint){
5         return addmod(14, 15, 13);
6     }
7     function callMulMod() public pure returns(uint){
8         return mulmod(14, 15,13);
9     }
}

```

ContractDefinition MathematicalFunction → 1 reference(s) ▾ ▾

[call] [call] from: 0x58380a6a701c568545dFcB03FcB8875f56beddC4 to: MathematicalFunction.(fallback) data: 0xd20...74110
call to MathematicalFunction.fallback

Output

The screenshot shows the Truffle UI interface. On the left, there's a sidebar titled "DEPLOY & RUN TRANSACTIONS" with options like "Display", "Publish to IPFS", and "At Address". Below it is a list of "Deployed Contracts" under "MATHEMATICALFUNCTION AT 0x7fb...". The main area shows the Solidity code for the MathematicalFunction contract:

```

pragma solidity ^0.5.0;

contract MathematicalFunction {
    function callAddMod() public pure returns(uint) {
        return addMod(14, 15, 13);
    }
    function callMulMod() public pure returns(uint) {
        return mulMod(14, 15, 23);
    }
}

```

On the right, the "ContractDefinition MathematicalFunction" pane shows "1 reference(s)". It lists three transactions recorded on the network:

- [call] From: 0x503...cc0c4 to: MathematicalFunction.(constructor) value: 0 wei data: 0x608...1002 logs: 0
- [call] From: 0x21...980d to: MathematicalFunction.callAddMod() data: 0xf5b...41691
- [call] From: 0x5300a60791...5685454cf089fc9875f56e0d4 to: MathematicalFunction.callMulMod() data: 0x64...e8744

// Cryptographic functions

```

function hashData(string memory data) public pure returns (bytes32) {
    return keccak256(abi.encodePacked(data));
}

```

The screenshot shows the Truffle UI interface. On the left, the "FILE EXPLORERS" sidebar shows a workspace named "default_workspace" containing several Solidity files: Variable.sol, Arrays.sol, Enums.sol, Struct.sol, Mapping.sol.sol, SpecialVariable.sol, whileLoop.sol, doWhileLoop.sol, Operators.sol, functionModifier.sol, ViewFunction.sol, PureFunction.sol, FallbackFunction.sol, FunctionOverloading.sol, MathematicalFunction.sol, and CryptographicFunction.sol. The "CryptographicFunction.sol" file is currently selected.

The main area shows the Solidity code for the CryptographicFunction contract:

```

pragma solidity ^0.5.0;

contract CryptographicFunction {
    function callKeccak256() public pure returns(bytes32 result) {
        return keccak256("ABC");
    }
}

```

Output:

The screenshot shows a blockchain development environment with the following details:

- Deploy & Run Transactions:** A sidebar on the left where a value of 0 Wei is selected for a deployment.
- Contract:** The contract named "CryptographicFunction" is selected.
- Deploy:** An orange button to initiate the deployment process.
- Publish to IPFS:** An option to publish the contract to the IPFS network.
- Transactions recorded:** A section showing the history of transactions for the deployed contract.
- Deployed Contracts:** A list showing "CRYPTOGRAPHICFUNCTION AT 0x00...".
- ContractDefinition Test:** A tab showing the contract's definition and pending transactions.
- Logs:** A list of logs from the deployed contract:
 - [vm] from: 0x50...e0d4 to: CryptographicFunction.(constructor) value: 0 wei data: 0x00...10002 logs: 0 hash: 0x873...a9eb call to cryptographicfunction.constructor()
 - [call] from: 0x55330da6201...5f05454cf1b13f4875f56bedd4 to: CryptographicFunction.callKeccak256() data: 0xb04...a03e hash: 0x873...a9eb

PRACTICAL 3 C.

Aim: Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.

Contracts:

Code:

```
pragma solidity ^0.8.0;

contract HelloContract {

    string public message;
    address public owner;

    // Constructor runs once on deployment
    constructor(string memory initialMessage) {
        message = initialMessage;
        owner = msg.sender;
    }

    // Function to update the message
    function updateMessage(string memory newMessage) public {
        require(msg.sender == owner, "Only the owner can update the message");
        message = newMessage;
    }

    // View function to return the message (also available as public variable)
    function getMessage() public view returns (string memory) {
        return message;
    }
}
```

The image consists of two vertically stacked screenshots of the Truffle UI interface. Both screenshots show the same Solidity code for a file named `Contracts.sol`.

```

1 pragma solidity ^0.5.0;
2
3 contract C {
4     //private state variable
5     uint private data;
6
7     //public state variable
8     uint public info;
9
10    //constructor
11    constructor() public {
12        info = 10;
13    }
14    //private function
15    function increment(uint a) private pure returns(uint) { return a + 1; }
16
17    //public function
18    function updateData(uint a) public { data = a; }
19    function getData() public view returns(uint) { return data; }
20    function compute(uint a, uint b) internal pure returns (uint) { return a + b; }
21 }
22 //External Contract
contract D {
23     function readData() public returns(uint) {
24         C c = new C();
25         c.updateData(7);
26         return c.getData();
27     }
28 }
29 //Derived Contract
contract E is C {
30     uint private result;
31     C private c;
32
33     constructor() public {
34         c = new C();
35     }
36     function getComputedResult() public {
37         result = compute(3, 5);
38     }
39     function getResult() public view returns(uint) { return result; }
40     function getData() public view returns(uint) { return c.info(); }
41 }
42
43 ContractDefinition E ↗ 1 reference(s) ↘

```

The left screenshot shows the `Contracts.sol` file selected in the left sidebar under the "FILE EXPLORERS" section. The right screenshot shows the same code with the `E` contract definition highlighted.

Output:

This screenshot shows the Truffle UI's "DEPLOY & RUN TRANSACTIONS" tab. It displays the deployment of contracts and the execution of transactions.

Publish to IPFS OR **At Address**: `0x00`

Transactions recorded: 2

Deployed Contracts: CAT (0x00)

updateData (ContractDefinition E ↗ 1 reference(s) ↘)

- [call] from: 0x58380a6c701c168545aCfC083Fc8875f56e0d4 to: C.getData() data: 0x30...5c30 Debug
- [call] from: 0x58380a6c701c168545aCfC083Fc8875f56e0d4 to: C.info() data: 0x30...158e Debug
- [vm] from: 0x0000...ed0E4 to: C.getComputedResult() data: 0x00...00007 logs: 0 Debug
- [call] from: 0x58380a6c701c168545aCfC083Fc8875f56e0d4 to: C.getData() data: 0x30...5c30 Debug

Low level interactions: Fail fast

Inheritance:**Code:**

```
pragma solidity ^0.8.0;

/// simple inheritance example with animals

/// @notice Base contract representing a generic animal
contract Animal {
    string public name;
    constructor(string memory _name) {
        name = _name;
    }

    function makeSound() public virtual pure returns (string memory) {
        return "Some generic animal sound";
    }

    function getName() public view returns (string memory) {
        return name;
    }
}

/// @notice Derived contract representing a dog
contract Dog is Animal {

    constructor(string memory _name) Animal(_name) {}

    // Overrides makeSound from Animal
    function makeSound() public pure override returns (string memory) {
        return "Woof!";
    }
}
```

```

function fetch() public pure returns (string memory) {
    return "Dog is fetching!";
}

}

```

```

/// @notice Derived contract representing a cat

contract Cat is Animal {

    constructor(string memory _name) Animal(_name) {}

    // Overrides makeSound from Animal

    function makeSound() public pure override returns (string memory) {
        return "Meow!";
    }

    function scratch() public pure returns (string memory) {
        return "Cat is scratching!";
    }
}

```

The screenshot shows the Truffle IDE interface with the Contracts.sol file open in the editor. The code defines a new contract, Cat, which inherits from the Animal contract. It overrides the makeSound() function and adds a new scratch() function that returns the string "Cat is scratching!". The editor interface includes a file explorer on the left, a toolbar at the top, and a status bar at the bottom.

```

FILE EXPLORERS
functionModifier.sol
ViewFunction.sol
PureFunction.sol
FallbackFunction.sol
FunctionOverloading.sol
MathematicalFunction.sol
CryptographyFunction.sol
Function.sol
String.sol
ArithmeticOperators.sol
RelationalOperator.sol
LogicalOperator.sol
BitwiseOperator.sol
AssignmentOperator.sol
ForLoop.sol
IfStatement.sol
ElseStatement.sol
IfElseStatement.sol
WithdrawPattern.sol
AccessControlContract.sol
Contracts.sol
Inheritance.sol
AbstractConstructor.sol
Constructor.sol
Events.sol
ErrorHandling.sol
scripts
tests
dops

```

```

pragma solidity >=0.5.0;

contract C {
    //private state variable
    uint private data;
    //public state variable
    uint public info;
    //constructor
    constructor() public {
        info = 20;
    }
    //private function
    function increment(uint a) private pure returns(uint) { return a + 1; }
    //public function
    function updateData(uint a) public { data = a; }
    function getData() public view returns(uint) { return data; }
    function compute(uint a, uint b) internal pure returns (uint) { return a + b; }
}
//Derived contract
contract E is C {
    uint private result;
    C private c;
    constructor() public {

```

```

21  */
22  //Derived Contract
23  Contract E is C {
24      uint private result;
25      C private c;
26      constructor() public {
27          c = new C();
28      }
29      function getComputedResult() public view returns(uint) { return result; }
30      function getResult() public view returns(uint) { return c.info(); }
31  }
32
33 }
```

Output:

```

5 uint private data;
6
7 //public state variable
8 uint public info;
9
10 //constructor
11 constructor() public {
12     info = 20;
13 }
14 //private function
15 function increment(uint a) private pure returns(uint) { return a + 1; }
16
17 //public function
```

Deployed Contracts: C AT 0x2E7... (MEMORY)

- updateData: 37
- getData: 0x00256...17
- info: 0x00256...20

Low level interactions: CALLDATA

Constructors:

Code:

```

pragma solidity ^0.8.0;

/// @title Demonstrates constructors in Solidity
/// @notice Base contract with a constructor

contract Person {

    string public name;

    uint public age;

    // Constructor to initialize name and age

    constructor(string memory _name, uint _age) {
```

```
name = _name;
age = _age;
}

function getDetails() public view returns (string memory, uint) {
    return (name, age);
}

}

/// @notice Derived contract that inherits Person and calls its constructor
contract Employee is Person {

    uint public employeeId;
    string public department;
    // Constructor to initialize inherited and new properties
    constructor(
        string memory _name,
        uint _age,
        uint _employeeId,
        string memory _department
    ) Person(_name, _age) {
        employeeId = _employeeId;
        department = _department;
    }

    function getEmployeeDetails() public view returns (
        string memory,
        uint,
        uint,
        string memory
    ) {
}
```

```

        return (name, age, employeeId, department);
    }
}

```

The screenshot shows the Truffle IDE interface. On the left, the 'FILE EXPLORERS' sidebar lists various Solidity files. In the main editor area, the code for 'Constructor.sol' is displayed:

```

pragma solidity ^0.5.0;
// Creating a contract
contract constructorExample {
    // Declaring state variable
    string str;

    // Creating a constructor
    // to set value of 'str'
    constructor() public {
        str = "This is Example of Constructor";
    }

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

```

Output:

The screenshot shows the 'DEPLOY & RUN TRANSACTIONS' tab in the Truffle IDE. It displays the deployment process for the 'constructorExample' contract. The code is identical to the one shown in the previous screenshot.

Deployment steps shown in the UI:

- Contract: constructorExample - contracts/Con...
- Deploy button is highlighted.
- Publish to IPFS checkbox is unchecked.
- At Address dropdown is selected.
- Transactions recorded: 27
- Deployed Contracts: CONSTRUCTOREXAMPLE AT 0x5E...

Contract interactions shown in the UI:

- getValue() function call results in: 0. string: This is Example of Constructor
- Low level interactions section shows a pending transaction for 'call to constructorExample.getValue()'.

Abstract Contracts:

Code:

```

pragma solidity ^0.8.0;

/// @title Abstract Contract Example: Shape
/// @notice Abstract contract defining a shape

```

```
abstract contract Shape {  
    string public shapeType;  
    constructor(string memory _shapeType) {  
        shapeType = _shapeType;  
    }  
    // Abstract function (no implementation)  
    function area() public view virtual returns (uint);  
}  
  
/// @notice Rectangle contract implementing Shape  
contract Rectangle is Shape {  
    uint public width;  
    uint public height;  
  
    constructor(uint _width, uint _height) Shape("Rectangle") {  
        width = _width;  
        height = _height;  
    }  
    function area() public view override returns (uint) {  
        return width * height;  
    }  
}  
  
/// @notice Circle contract implementing Shape  
contract Circle is Shape {  
    uint public radius;  
    constructor(uint _radius) Shape("Circle") {  
        radius = _radius;  
    }  
}
```

```
function area() public view override returns (uint) {
    // Approximate π as 314 / 100
    return (314 * radius * radius) / 100;
}
```

The screenshot shows the Truffle UI interface. On the left, the 'FILE EXPLORERS' sidebar lists various Solidity files: doWhileLoop.sol, functionModifiers.sol, ViewFunction.sol, PureFunction.sol, FallBackFunction.sol, FunctionOverloading.sol, MathematicalFunction.sol, CryptographicFunction.sol, Function.sol, string.sol, ArithmeticOperations.sol, RelationalOperator.sol, LogicalOperator.sol, BitwiseOperator.sol, AssignmentOperator.sol, forLoop.sol, ifStatement.sol, ifElseStatement.sol, withdrawalPattern.sol, accessRestriction.sol, Contracts.sol, Inheritance.sol, abstractConstructor.sol (selected), Constructor.sol, Events.sol, ErrorHandling.sol, scripts, tests, and migrations. The main panel displays the contents of abstractConstructor.sol and inheritance.sol. abstractConstructor.sol contains a pragma solidity ^0.5.0; statement and two contracts: abstractConstructor and Test. Test inherits from abstractConstructor and defines a getResult() function that adds two uint variables (a and b) and returns the result. inheritance.sol defines a base contract with a single function, which is then overridden by the Test contract.

```
abstractConstructor.sol
pragma solidity ^0.5.0;
contract abstractConstructor {
    function getResult() public view returns(uint);
}
contract Test is abstractConstructor {
    function getResult() public view returns(uint) {
        uint a = 10;
        uint b = 17;
        uint result = a + b;
        return result;
    }
}

inheritance.sol
contract Base {
    function getResult() public view returns(uint);
}
contract Test is Base {
    function getResult() public view returns(uint) {
        uint a = 10;
        uint b = 17;
        uint result = a + b;
        return result;
    }
}
```

Output:

The screenshot shows the Truffle UI interface under the 'DEPLOY & RUN TRANSACTIONS' tab. It displays a deployment interface for the Test contract, showing options to 'Deploy' or 'Publish to IPFS'. Below this, it shows 'Transactions recorded' and 'Deployed Contracts'. A specific transaction for 'getArea()' is highlighted, showing its details: from: 0x5B3...c60C to: Test-(constructor) value: 0 wei data: 0x600...10002 logs: 0 hash: 8xf7...96e41. The bottom section shows the 'Low level interactions' for the TEST-AT-0X5B3...C60C (MEMORY) contract, with a 'getArea()' button and a 'Debug' button.

Interface:

Code:

```
pragma solidity ^0.8.0;

/// @title Demonstrates interface in Solidity

/// @notice Interface defining a calculator

interface ICalculator {

    function add(uint a, uint b) external pure returns (uint);

    function subtract(uint a, uint b) external pure returns (uint);

    function multiply(uint a, uint b) external pure returns (uint);

    function divide(uint a, uint b) external pure returns (uint);

}

/// @notice Implementation of the calculator interface

contract SimpleCalculator is ICalculator {

    function add(uint a, uint b) external pure override returns (uint) {

        return a + b;

    }

    function subtract(uint a, uint b) external pure override returns (uint) {

        require(a >= b, "Underflow error");

        return a - b;

    }

    function multiply(uint a, uint b) external pure override returns (uint) {

        return a * b;

    }

    function divide(uint a, uint b) external pure override returns (uint) {

        require(b != 0, "Division by zero");

        return a / b;

    }

}

/// @notice A contract that uses the ICalculator interface
```

```

contract CalculatorUser {
    ICalculator public calculator;
    constructor(address _calculatorAddress) {
        calculator = ICalculator(_calculatorAddress);
    }
    function useCalculator(uint a, uint b) public view returns (uint sum, uint diff, uint prod, uint quot) {
        sum = calculator.add(a, b);
        diff = calculator.subtract(a, b);
        prod = calculator.multiply(a, b);
        quot = calculator.divide(a, b);
    }
}

```

```

FILE EXPLORERS
PureFunction.sol
FallBackFunction.sol
FunctionOverriding.sol
MathematicalFunction.sol
CryptographicFunction.sol
Function.sol
String.sol
ArithmeticOperators.sol
RelationalOperator.sol
LogicalOperator.sol
BitwiseOperator.sol
AssignmentOperator.sol
ForLoop.sol
Statement.sol
IfElseStatement.sol
IfElseIfStatement.sol
withdrawalPattern.sol

Interface.sol
pragma solidity ^0.5.0;

contract Interface {
    function getResult() public view returns(uint);
}

contract Test is Interface {
    function getResult() public view returns(uint) {
        uint a = 31;
        uint b = 67;
        uint result = a + b;
        return result;
    }
}

```

Output:

```

DEPLOY & RUN TRANSACTIONS
CONTRACT
Test - contracts/Interface.sol
Deploy
Publish to IPFS
OR
At Address
Transactions recorded 23
Deployed Contracts
TEST AT 0x5E303De781c360545dCfC987f596e0dA4 to: Test.getResult() data: 0x00...02709
getResult
0.00000000 0
Low level interactions
CALLDATA
Truffle

```

```

Interface.sol
pragma solidity ^0.5.0;

contract Interface {
    function getResult() public view returns(uint);
}

contract Test is Interface {
    function getResult() public view returns(uint) {
        uint a = 31;
        uint b = 67;
        uint result = a + b;
        return result;
    }
}

```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with options like 'Deploy', 'Publish to IPFS', and 'At Address'. Below that is a section for 'Transactions recorded' with 23 entries. A list of 'Deployed Contracts' shows 'TEST AT 0x5E303De781c360545dCfC987f596e0dA4'. Under 'Low level interactions', there's a 'CALLDATA' section with a 'Truffle' button.

PRACTICAL 3 D.

Aim: Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling

Libraries:

Code:

```
pragma solidity ^0.8.0;

/// @title Demonstrates use of libraries in Solidity
/// @notice A simple math library

library MathLib {

    function add(uint a, uint b) internal pure returns (uint) {
        return a + b;
    }

    function subtract(uint a, uint b) internal pure returns (uint) {
        require(a >= b, "Underflow error");
        return a - b;
    }

    function multiply(uint a, uint b) internal pure returns (uint) {
        return a * b;
    }

    function divide(uint a, uint b) internal pure returns (uint) {
        require(b != 0, "Division by zero");
        return a / b;
    }
}

/// @notice A contract that uses the MathLib library
contract Calculator {
```

```

using MathLib for uint;

function compute(
    uint a,
    uint b
) public pure returns (uint sum, uint diff, uint prod, uint quot) {
    sum = a.add(b);
    diff = a.subtract(b);
    prod = a.multiply(b);
    quot = a.divide(b);
}
}

```

The screenshot shows the Solidity IDE interface. On the left, the 'FILE EXPLORERS' sidebar lists various Solidity files such as FallbackFunction.sol, FunctionOverloading.sol, MathematicalFunction.sol, CryptographicFunction.sol, FunctionTest.sol, string.sol, ArithmeticOperators.sol, RelationalOperator.sol, LogicalOperator.sol, BitwiseOperator.sol, AssignmentOperator.sol, forLoop.sol, IfStatement.sol, IfElseStatement.sol, IfElseIfStatement.sol, withdrawalPattern.sol, accessRestriction.sol, Contracts.sol, Inheritance.sol, abstractConstructor.sol, Constructor.sol, Events.sol, ErrorHandling.sol, Interface.sol, and Library.sol. The 'Library.sol' file is currently selected in the sidebar. The main editor area displays the Solidity code for the 'Library' contract, which includes a library named 'Search' and a contract named 'library'. The code implements a search function that iterates through an array to find a specific value.

```

pragma solidity ^0.5.6;

library Search {
    function indexOf(uint[] storage self, uint value) public view returns (uint) {
        for (uint i = 0; i < self.length; i++) if (self[i] == value) return i;
    }
}

contract library {
    uint[] data;
    constructor() public {
        data.push(1);
        data.push(2);
        data.push(3);
        data.push(4);
        data.push(5);
    }
    function isValuePresent() external view returns (bool) {
        uint value = 4;
        //search if value is present in the array using library function
        uint index = Search.indexOf(data, value);
        return index;
    }
}

```

Output:

```

library Library {
    uint[] data;

    function push(uint value) external {
        data.push(value);
    }

    function isValuePresent() external view returns(uint) {
        uint value = 4;
        //search if value is present in the array.using library function
        uint index = Search.indexOf(data, value);
        return index;
    }
}

```

Assembly:

Code:

```

pragma solidity ^0.8.0;

/// @title Demonstrates inline assembly usage in Solidity

contract AssemblyDemo {

    /// @notice Adds two numbers using assembly
    function addAsm(uint a, uint b) public pure returns (uint result) {

        assembly {
            result := add(a, b)
        }
    }

    /// @notice Returns caller address using assembly
    function getCaller() public view returns (address caller) {

        assembly {
            caller := caller()
        }
    }
}

```

```
}

/// @notice Returns the square of a number using multiplication in assembly
function square(uint x) public pure returns (uint result) {

    assembly {
        result := mul(x, x)
    }
}

/// @notice Demonstrates conditional branching with assembly
function isEven(uint number) public pure returns (bool even) {

    assembly {
        switch mod(number, 2)
        case 0 {
            even := 1
        }
        default {
            even := 0
        }
    }
}
```

The screenshot shows the Solidity IDE interface with the following details:

- FILE EXPLORERS**: On the left, a sidebar lists various Solidity files: FunctionOverloading.sol, MathematicalFunction.sol, CryptographicFunction.sol, Function.sol, string.sol, ArithmeticOperators.sol, RelationalOperator.sol, LogicalOperator.sol, BitwiseOperator.sol, AssignmentOperator.sol, forLoop.sol, ifStatement.sol, ifElseStatement.sol, withdrawalPattern.sol, accessRestriction.sol, Contracts.sol, Inheritance.sol, abstractConstructor.sol, Constructor.sol, Events.sol, ErrorHandling.sol, Interface.sol, Library.sol, and Assembly.sol.
- Contract Definition**: The current file is `Assembly.sol`, which contains the following Solidity code:

```
pragma solidity >0.4.8;

contract Assembly {

    function add(uint a) view returns (uint b) {
        assembly {
            let c := add(a, 16)
            mstore(0x00, c)
            {
                let d := add(sload(c), 12)
                // assign the value of 'd' to 'b'
                b := d
                // 'd' is deallocated now
            }
            b := add(b, c)
        }
    }
}
```
- Toolbars and Status Bar**: At the bottom, there are toolbars for navigation, search, and transaction monitoring. The status bar displays "ContractDefinition Assembly" and "1 references".

Output:

```
pragma solidity ^0.4.8;
contract Assembly {
    function add(uint a) view returns (uint b) {
        assembly {
            let c := add(a, 16)
            mstore(0x80, c)
            let d := add(sload(c, 12))
            // assign the value of 'd' to 'b'
        }
    }
}
```

Events:

Code:

```
pragma solidity ^0.8.0;
```

```
/// @title Demonstrates the use of events in Solidity
```

```
contract EventDemo {
```

```
// Declare an event that logs when a user registers
```

```
event UserRegistered(address indexed userAddress, string username);
```

```
// Declare an event for balance updates
```

```
event BalanceUpdated(address indexed user, uint newBalance);

mapping(address => uint) public balances;

mapping(address => string) public usernames;

/// @notice Register a new user with a username

function register(string calldata _username) external {

    require(bytes(usernames[msg.sender]).length == 0, "User already registered");

    usernames[msg.sender] = _username;

    // Emit the UserRegistered event

    emit UserRegistered(msg.sender, _username);

}

/// @notice Deposit some amount to the sender's balance

function deposit() external payable {

    require(msg.value > 0, "Must send some ether");

    balances[msg.sender] += msg.value;

    // Emit the BalanceUpdated event

    emit BalanceUpdated(msg.sender, balances[msg.sender]);

}

/// @notice Withdraw a specified amount from sender's balance

function withdraw(uint _amount) external {

    require(balances[msg.sender] >= _amount, "Insufficient balance");

    balances[msg.sender] -= _amount;

    payable(msg.sender).transfer(_amount);

    // Emit the BalanceUpdated event

    emit BalanceUpdated(msg.sender, balances[msg.sender]);

}

}
```

The screenshot shows the Truffle UI interface. On the left, the 'FILE EXPLORERS' sidebar lists various Solidity files: FunctionOverloading.sol, MathematicalFunction.sol, CryptographicFunction.sol, Function.sol, string.sol, ArithmeticOperators.sol, RelationalOperator.sol, LogicalOperator.sol, BitwiseOperator.sol, AssignmentOperator.sol, forLoop.sol, ifStatement.sol, ifElseStatement.sol, withdrawalPattern.sol, accessRestriction.sol, Contracts.sol, Inheritance.sol, abstractConstructor.sol, Constructor.sol, Events.sol (which is selected), ErrorHandling.sol, Interface.sol, Library.sol, Assembly.sol, scripts, tests, and deps. At the bottom of the sidebar is a README.txt file. The main area is a code editor with the 'Events.sol' file open. The code defines a contract 'Events' with a state variable 'value' and an event 'Increment'. It also includes a function 'getValue' that logs the value to the event.

```

2: // creating an event
3: pragma solidity ^0.4.21;
4:
5: // Creating a contract
6: contract Events {
7:
8:     // Declaring state variables
9:     uint256 public value = 0;
10:
11:    // Declaring an event
12:    event Increment(address owner);
13:
14:    // Defining a function for logging event
15:    function getValue(uint _a, uint _b) public {
16:        emit Increment(msg.sender);
17:        value = _a + _b;
18:    }
19:
20}

```

Output:

The screenshot shows the Truffle UI interface with the 'DEPLOY & RUN TRANSACTIONS' sidebar active. It displays a 'Deploy' button, a 'Publish to IPFS' option, and a dropdown for selecting a deployment address ('At Address'). Below these are sections for 'Transactions recorded' (45), 'Deployed Contracts', and 'EVENTS AT 0x628...5503 memory'. The 'Events' contract is selected, showing its source code and event definitions. A transaction history table is shown, with one entry for a pending transaction to 'Events.getvalue' from address 0x03E3...e6dC4 to the contract's address 0xf26...35019. The table includes columns for tx hash, from, to, value, data, logs, and a 'Debug' button. At the bottom, there are sections for 'Low level interactions' and 'Contract details'.

Error Handling:

Code:

```
pragma solidity ^0.8.0;

/// @title Demonstrates error handling in Solidity

contract ErrorHandlingDemo {

    mapping(address => uint) public balances;

    /// @notice Deposit ether into your balance

    function deposit() external payable {

        require(msg.value > 0, "Deposit must be greater than zero");

        balances[msg.sender] += msg.value;

    }

    /// @notice Withdraw specified amount of ether

    function withdraw(uint _amount) external {

        // Use revert with custom error message

        if (_amount > balances[msg.sender]) {

            revert("Insufficient balance");

        }

        balances[msg.sender] -= _amount;

        // Send ether back to caller

        (bool success, ) = msg.sender.call{value: _amount}("");

        require(success, "Transfer failed");

    }

    function testAssert(uint _value) external pure {

        // Assert should be used for conditions that should never fail

        assert(_value != 0);

    }

}
```

The screenshot shows the Truffle IDE interface. The left sidebar, titled "FILE EXPLORERS", lists several Solidity files: FunctionOverriding.sol, MathematicalFunction.sol, CryptographicFunction.sol, Function.sol, string.sol, ArithmeticOperators.sol, RelationalOperator.sol, LogicalOperator.sol, BitwiseOperator.sol, AssignmentOperator.sol, forLoop.sol, #Statement.sol, #ForStatement.sol, #IfStatement.sol, #WhileStatement.sol, withdrawPattern.sol, accessRestrictions.sol, Contracts.sol, Inheritance.sol, abstractConstructor.sol, Constructor.sol, Events.sol, ErrorHandling.sol (which is currently selected), Interface.sol, Library.sol, Assembly.sol, scripts, tests, deps, and README.txt.

The main right-hand area is a code editor for the file "ErrorHandling.sol". The code contains Solidity contracts and functions:

```
pragma solidity ^0.5.6;

contract ErrorHandling {

    function checkInput(uint _input) public view returns(string memory) {
        require(_input >= 0, "invalid uint8");
        require(_input <= 255, "invalid uint8");

        return "Input is uint8";
    }

    // Defining function to use require statement.
    function Odd(uint _input) public view returns(bool) {
        require(_input % 2 != 0);
        return true;
    }
}
```

Output:

```
pragma solidity ^0.5.8;

contract ErrorHandling {

    function checkInput(uint _input) public view returns(string memory) {
        require(_input >= 0, "Invalid uint8");
        require(_input <= 255, "Invalid uint8");

        return "Input is Uint8";
    }

    // Defining function to use require statement
    function Odd(uint _input) public view returns(bool) {
        require(_input % 2 != 0);
        return true;
    }
}
```

ContractDefinition ErrorHandling 1 reference(s)

call to errorhandling.checkInput

[call] From: 0x5B38906a701c568545dCfc883Fc8875f58edc4 to: ErrorHandling.checkInput(uint256) data: 0x81...000f3 Debug

call to errorhandling.odd

[call] From: 0x5B38906a701c568545dCfc883Fc8875f58edc4 to: ErrorHandling.odd(uint256) data: 0x82...000f3 Debug

PRACTICAL 3 E.

Aim: Build a decentralized application (DApp) using Angular for the front end and Truffle along with Ganache CLI for the back end.

Code:

To build a decentralized application (DApp) using Angular for the front end and Truffle with Ganache CLI for the back end, there are following steps

1. **Set up Ganache CLI** (for local Ethereum blockchain)
2. **Create and deploy a smart contract using Truffle**
3. **Build the Angular front end**
4. **Integrate Angular front end with Ethereum blockchain**

Step 1: Set up Ganache CLI (Ethereum Local Blockchain)

First, install Ganache CLI. Ganache is a personal Ethereum blockchain which you can use for development purposes.

1. Install Ganache CLI globally using npm:
2. `npm install -g ganache-cli`
3. Run Ganache CLI to start a local Ethereum blockchain:
4. `ganache-cli`

By default, this will run on `http://127.0.0.1:8545` and you'll see the list of accounts with balances.

Step 2: Create and Deploy Smart Contract with Truffle

1. Install Truffle

Install Truffle globally on your machine:

```
npm install -g truffle
```

2. Initialize a Truffle Project

Create a directory for your project, then initialize Truffle:

```
mkdir my-dapp
cd my-dapp
truffle init
```

3. Write a Smart Contract

In the contracts/ folder, create a file SimpleStorage.sol with a simple smart contract to store and retrieve a number.

```
// contracts/SimpleStorage.sol
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 private storedNumber;

    function set(uint256 num) public {
        storedNumber = num;
    }

    function get() public view returns (uint256) {
        return storedNumber;
    }
}
```

4. Compile the Contract

In the project directory, run:

```
truffle compile
```

5. Deploy the Contract

Create a migration file in migrations/2_deploy_contracts.js:

```
const SimpleStorage = artifacts.require("SimpleStorage");

module.exports = function (deployer) {
    deployer.deploy(SimpleStorage);
};
```

Now, deploy the contract to your local Ganache instance:

```
truffle migrate --network development
```

6. Interact with the Contract

To test interactions, run the Truffle console:

```
truffle console --network development
```

In the console, interact with the deployed contract:

```
let instance = await SimpleStorage.deployed();
await instance.set(42); // Set a value
let value = await instance.get(); // Get the stored value
console.log(value.toString()); // Should print 42
```

Step 3: Build the Angular Front End

1. Create Angular Project

Create an Angular project using the Angular CLI:

```
ng new my-dapp-frontend
cd my-dapp-frontend
```

Install the necessary dependencies for Web3.js (to interact with Ethereum):

```
npm install web3
```

2. Create a Service to Interact with the Blockchain

In your Angular project, create a service to connect to the blockchain and interact with the smart contract.

Generate a service:

```
ng generate service blockchain
```

Edit blockchain.service.ts to connect with Ganache and interact with the SimpleStorage contract:

```
import { Injectable } from '@angular/core';
import Web3 from 'web3';
import { environment } from './environments/environment';

@Injectable({
  providedIn: 'root'
})
export class BlockchainService {

  private web3: Web3;
```

```

private contract: any;
private contractAddress = 'YOUR_CONTRACT_ADDRESS';
private contractABI = [ /* ABI from Truffle build */ ];

constructor() {
  this.web3 = new Web3('http://127.0.0.1:8545'); // Connect to Ganache CLI
  this.contract = new this.web3.eth.Contract(this.contractABI, this.contractAddress);
}

async getStoredNumber(): Promise<number> {
  return await this.contract.methods.get().call();
}

async setStoredNumber(number: number): Promise<void> {
  const accounts = await this.web3.eth.getAccounts();
  await this.contract.methods.set(number).send({ from: accounts[0] });
}

```

3. Display Data in the Component

```

import { Component } from '@angular/core';
import { BlockchainService } from './blockchain.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'My DApp';
  storedNumber: number;

  constructor(private blockchainService: BlockchainService) {
    this.loadStoredNumber();
  }

  async loadStoredNumber() {
    this.storedNumber = await this.blockchainService.getStoredNumber();
  }

  async setStoredNumber(number: number) {
    await this.blockchainService.setStoredNumber(number);
    this.loadStoredNumber(); // Refresh the number
  }
}

```

4. Update the Template

Update the app.component.html to display the stored number and provide an input to set a new number:

```
<div style="text-align:center;">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <p>The current stored number is: {{ storedNumber }}</p>
  <input type="number" [(ngModel)]="newNumber" placeholder="Enter number" />
  <button (click)="setStoredNumber(newNumber)">Set Number</button>
</div>
```

5. Add Angular Forms Module

In app.module.ts, import the necessary module:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { BlockchainService } from './blockchain.service';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, FormsModule],
  providers: [BlockchainService],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Step 4: Run the Application

1. **Start Ganache CLI** (if not already running):
2. ganache-cli
3. **Deploy the smart contract** (if not already deployed):
4. truffle migrate --network development
5. **Run the Angular Application:**
6. ng serve

Now, visit <http://localhost:4200/> in your browser. You should see the stored number from your smart contract, and you can set a new number using the input field.

PRACTICAL 4 A.

Aim: Install and demonstrate use of hyperledger-Iroha

Code:

Below is a step-by-step guide to **install Hyperledger Iroha** and demonstrate its basic functionality.

Step 1: Prerequisites

1. **Docker:** Hyperledger Iroha uses Docker containers for running its components.
 - o Install Docker from [Docker's official site](#).
2. **Docker Compose:** This tool allows you to define and run multi-container Docker applications.
 - o Install Docker Compose from [here](#).
3. **Git:** To clone repositories.
 - o Install Git from [here](#).
4. **Java:** Hyperledger Iroha is developed using Java, so make sure you have Java (JDK 11 or higher) installed.
 - o Install Java from [here](#).
5. **CMake and build tools** (for building the Hyperledger Iroha binaries, if you plan to build it yourself):
 - o Install CMake from [here](#).
 - o Install build-essential (for Linux) via the package manager:
 - o `sudo apt-get install build-essential`

Step 2: Clone Hyperledger Iroha Repository

Clone the Hyperledger Iroha repository from GitHub to your local machine.

```
git clone https://github.com/hyperledger/iroha.git
cd iroha
```

Step 3: Build Hyperledger Iroha (Optional Step)

If you want to build Hyperledger Iroha from source, follow these steps:

1. **Install dependencies:**
 - o For Ubuntu:
 - o `sudo apt-get update`
 - o `sudo apt-get install -y cmake g++ libssl-dev libboost-all-dev \`
 - o `libgmp-dev libsodium-dev python3-pip libcurl4-openssl-dev \`
 - o `zlib1g-dev libzmq3-dev liblzma-dev libprotobuf-dev protobuf-compiler`

- For macOS, use Homebrew to install dependencies:
 - brew install cmake boost openssl libsodium libzmq protobuf
2. **Build the project:**
 3. mkdir build
 4. cd build
 5. cmake ..
 6. make -j\$(nproc) # Use appropriate number of CPU cores for faster build

This step builds Hyperledger Iroha from source. If you just want to run the demo (using pre-built binaries), you can skip this step and proceed to using Docker.

Step 4: Use Docker to Run Hyperledger Iroha

Instead of building from source, we can use Docker to easily spin up the Iroha network.

1. **Use Docker Compose to bring up the containers:**

In the iroha directory, you should find a docker-compose.yml file. To start the Iroha network with all the components (like the peer, validator, etc.), run the following command:

2. docker-compose -f docker/docker-compose.yml up

This will pull the necessary images and start Iroha in Docker containers. You'll see logs of various services being initialized.

3. **Check if everything is up:**

After a few seconds, check if the containers are running:

4. docker ps

You should see containers for iroha and its components such as iroha-peer, iroha-ledger, etc.

Step 5: Interact with the Iroha Network

Once the containers are running, you can interact with the network. Hyperledger Iroha provides a REST API and a JavaScript client SDK (IrohaJS), but for simplicity, we'll start by interacting using curl or any HTTP client.

1. Create a User

To interact with the Iroha blockchain, you need to create a user. Iroha uses commands like create account, add asset, etc. Let's start by creating a user using the REST API.

- To create an account, we can send a request to the REST API like so (you may want to modify it according to your setup, e.g., IP and port):

```
curl -X POST http://localhost:50051/create_account \
-H "Content-Type: application/json" \
-d '{
    "creator": "admin",
    "name": "user1",
    "password": "password123"
}'
```

This will create an account named user1 with the password password123 under the admin account.

2. View Account Information

To verify the creation of the account and view account details, send a get account command.

```
curl -X GET http://localhost:50051/account/user1
```

This should return the details of the newly created account.

3. Transfer Assets Between Accounts

Now that we have two accounts, we can transfer assets between them. Let's transfer a certain number of assets from one account to another:

```
curl -X POST http://localhost:50051/transfer \
-H "Content-Type: application/json" \
-d '{
    "from": "user1",
    "to": "user2",
    "amount": "100",
    "asset_id": "coin"
}'
```

This will transfer 100 units of the asset coin from user1 to user2.

4. Check Transaction Status

You can also check the status of a transaction to ensure that it was successful. Here's how to do it:

```
curl -X GET http://localhost:50051/transaction_status \
-d '{"tx_id": "your-transaction-id"}'
```

Replace "your-transaction-id" with the actual ID of the transaction.

Step 6: Using Hyperledger Iroha SDK

You can use the Hyperledger Iroha SDK (available in several languages, such as Java, Python, and JavaScript) to interact with Iroha more programmatically.

Using IrohaJS

1. Install IrohaJS:

```
npm install iroha-helpers iroha-client
```

2. Interact with the Iroha network:

```
const { Iroha, IrohaAPI } = require('iroha-helpers');

// Create a connection to the Iroha network
const iroha = new Iroha('localhost', 50051);

// Create an account
iroha.createAccount('user1', 'password123').then(response => {
  console.log(response);
});

// Transfer assets
iroha.transferAsset('user1', 'user2', 'coin', 100).then(response => {
  console.log(response);
});
```

Step 7: Shutting Down Iroha

```
docker-compose -f docker/docker-compose.yml down
```

This will stop and remove the containers, freeing up resources.

Summary of Steps:

1. **Install prerequisites** like Docker, Java, and Git.
2. **Clone the Hyperledger Iroha repository** and optionally build it from source.
3. **Use Docker Compose** to set up Iroha and all its components.
4. **Interact with the blockchain** via HTTP API (curl) or SDK (like IrohaJS).
5. **Shut down the network** when done.

By following these steps, you've now set up Hyperledger Iroha and can interact with the permissioned blockchain for a simple, mobile-friendly blockchain application!

PRACTICAL 4 B.

Aim: Demonstration on interacting with NFT

Code:

Interacting with **Non-Fungible Tokens (NFTs)** using **Hyperledger Iroha** involves creating and managing digital assets that are unique and can be associated with specific metadata, such as artwork, collectibles, or other unique items.

Iroha has native support for **assets** (which can be created as NFTs). An asset in Hyperledger Iroha is essentially a unique, transferrable item that can be issued or moved between users.

Steps to Demonstrate NFT Interactions with Hyperledger Iroha:

1. **Create a New Asset (NFT):**
 - An asset in Iroha can be anything, but for NFTs, you would generally create an asset with a **unique ID** and **metadata** to distinguish it.
2. **Transfer the NFT:**
 - Transfer NFTs between users, which could be useful for trading or ownership transfer.
3. **View NFT Information:**
 - Retrieve metadata about the NFT, such as its ownership and other associated data.
4. **Use the Hyperledger Iroha REST API or Iroha SDKs** (like IrohaJS) to interact with the NFTs.

Step 1: Set Up Hyperledger Iroha

Follow the steps in the previous answer to **set up Hyperledger Iroha** using Docker. If you've already done that, you can skip to the next steps.

Step 2: Create a User and Define an NFT Asset

Once the Hyperledger Iroha network is up and running, you can interact with it using the Iroha REST API. To create an NFT, you first need to:

- **Create a user.**
- **Create a unique asset** for that user, which represents the NFT.

1. Create a User (user1)

First, create a user using the API:

```
curl -X POST http://localhost:50051/create_account \
-H "Content-Type: application/json" \
-d '{
    "creator": "admin",
    "name": "user1",
    "password": "password123"
}'
```

2. Create a Unique Asset (NFT)

You can define the NFT asset by creating a new asset that has a unique identifier (e.g., nft:1, nft:2, etc.).

Let's create an NFT for user1:

```
curl -X POST http://localhost:50051/create_asset \
-H "Content-Type: application/json" \
-d '{
    "creator": "admin",
    "account_id": "user1",
    "asset_id": "nft:1",
    "amount": "1",
    "description": "This is a unique digital collectible."
}'
```

This will create an asset nft:1 associated with user1, with an amount of 1 (since each NFT is unique, its amount is 1).

Note: In this case, we're just creating a basic "asset" with a unique asset_id that could represent an NFT. In a real NFT use case, the metadata might include things like images, links to digital artworks, or other relevant data. We can further extend this idea by associating more data with the NFT.

3. Add Metadata to the NFT (Optional)

Iroha allows you to associate metadata with assets. You can add metadata to your NFT to give it more details (like a link to the artwork).

```
curl -X POST http://localhost:50051/add_metadata \
-H "Content-Type: application/json" \
-d '{
    "account_id": "user1",
    "asset_id": "nft:1",
    "metadata": {
        "url": "https://example.com/nft1"
    }
}'
```

```
        "creator": "ArtistName",
        "image_url": "http://example.com/artwork1.png",
        "description": "A one-of-a-kind digital collectible."
    }
}
```

This would add metadata like the creator's name, an image URL, and a description to the NFT.

Step 3: Transfer the NFT Between Users

Once you've created an NFT, you can transfer it between users to simulate the buying, selling, or trading of NFTs.

Let's transfer the nft:1 asset from user1 to user2.

1. Create user2 (if not created already)

```
curl -X POST http://localhost:50051/create_account \
-H "Content-Type: application/json" \
-d '{
    "creator": "admin",
    "name": "user2",
    "password": "password123"
}'
```

2. Transfer the NFT

Now, you can transfer the NFT from user1 to user2.

```
curl -X POST http://localhost:50051/transfer \
-H "Content-Type: application/json" \
-d '{
    "from": "user1",
    "to": "user2",
    "amount": "1",
    "asset_id": "nft:1"
}'
```

This will transfer the nft:1 asset from user1 to user2.

Step 4: Verify the Transfer and View the NFT Information

You can check if the transfer was successful and see details about the NFT.

1. Check Account Information

After the transfer, you can check if user2 now owns the NFT (nft:1).

```
curl -X GET http://localhost:50051/account/user2
```

This will display the assets associated with user2, and you should see nft:1 listed there.

2. Retrieve Metadata for the NFT

To get the metadata for the NFT (such as the creator, image URL, and description), you can use the following command:

```
curl -X GET http://localhost:50051/metadata/nft:1
```

This should return the metadata associated with nft:1, including information like the image URL, creator, and description.

Step 5: Interact with NFTs Using IrohaJS

To interact programmatically with Iroha from a JavaScript-based frontend, you can use **IrohaJS**.

1. Install IrohaJS

You need to install the iroha-helpers package.

```
npm install iroha-helpers iroha-client
```

2. Write JavaScript Code to Create and Transfer NFTs

Here's an example of how to create and transfer an NFT programmatically:

```
const { Iroha, IrohaAPI } = require('iroha-helpers');

// Set up the connection to Iroha
const iroha = new Iroha('localhost', 50051);

// Create an account (if not already created)
async function createAccount() {
  const response = await iroha.createAccount('user1', 'password123');
  console.log('Account created:', response);
}

// Create an NFT (asset)
```

```

async function createNFT() {
  const nftResponse = await iroha.createAsset('user1', 'nft:1', '1', 'This is a unique digital
collectible');
  console.log('NFT created:', nftResponse);
}

// Transfer the NFT to another account
async function transferNFT() {
  const transferResponse = await iroha.transferAsset('user1', 'user2', 'nft:1', '1');
  console.log('NFT transferred:', transferResponse);
}

// Retrieve NFT metadata
async function getNFTMetadata() {
  const metadata = await iroha.getAssetMetadata('nft:1');
  console.log('NFT metadata:', metadata);
}

// Run the functions
createAccount().then(() => createNFT()).then(() => transferNFT()).then(() => getNFTMetadata());

```

Step 6: Shutting Down Iroha

After you're done interacting with Hyperledger Iroha, stop the containers:

```
docker-compose -f docker/docker-compose.yml down
```

Summary of NFT Interactions

1. **Created NFT:** We defined an asset (e.g., nft:1) as a unique NFT for user1.
2. **Added Metadata:** We added metadata like creator and image URL to make the NFT more descriptive.
3. **Transferred NFT:** We demonstrated transferring the NFT from user1 to user2.
4. **Used IrohaJS:** We showed how to interact with Iroha programmatically via JavaScript.

This simple demonstration provides a solid foundation for building more complex applications that involve NFTs, such as digital art platforms or unique asset management systems using Hyperledger Iroha.