NURTURING POTENTIAL

SAKET GYANPEETH'S

SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE

KALYAN (EAST)

ACADEMIC YEAR 2024-25

M.Sc. Information Technology

Part II NEP 2020 Semester IV

SUBMITTED BY

Mr. SACHIDANAND SANTOSH DUBEY

AS PRESCRIBED BY

UNIVERSITY OF MUMBAI

MUMBAI UNIVERSITY

NURTURING POTENTIAL

SAKET GYANPEETH'S

SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Permanently Affiliated to University of Mumbai)

NAAC Accredited

Saket Vidyanagri Marg, Chinchpada Road, Katemanivali,
Kalyan (East) -421306(Mah)

Department of Information Technology

This is to certify that

Mr. SACHIDANAND SANTOSH DUBEY

Seat No. 1314296

of

M.Sc. Information Technology

Part II NEP 2020 Semester IV

has satisfactory carried out the required practical in the subject

of **BLOCKCHAIN**

For the Academic year 2024 – 2025

Practical In-Charge **Head of the Department**

External Examiner

College Seal

INDEX

Sr. No.		Practical	Signature
1.	a.	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.	
	b.	Allow users to create multiple transactions and display them in an organised format	
	c.	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	
	d.	Implement a function to add new blocks to the miner and dump the blockchain	
2.	a.	Write a python program to demonstrate mining.	
	b.	Demonstrate the use of the Bitcoin Core API to interact with a Bitcoin Core node.	
	c.	Demonstrating the process of running a blockchain node on your local machine	
	d.	Demonstrate mining using geth on your private network.	

3.	a.	Write a Solidity program that demonstrates various types of functions including regular functions, view functions, pure functions, and the fallback function.	
	b.	Write a Solidity program that demonstrates function overloading, mathematical functions, and cryptographic functions.	
	c.	Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.	
	d.	Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling	
	e.	Build a decentralized application (DApp) using Angular for the front end and Truffle along with Ganache CLI for the back end.	
4.	a.	Install and demonstrate use of hyperledger-Irhoa	
	b.	Demonstration on interacting with NFT	

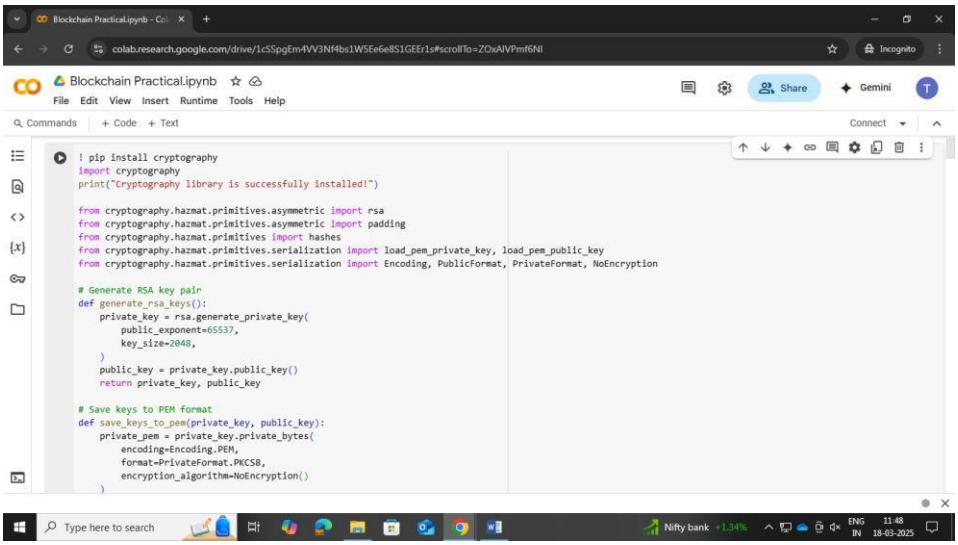
PRACTICAL 1 A.

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.

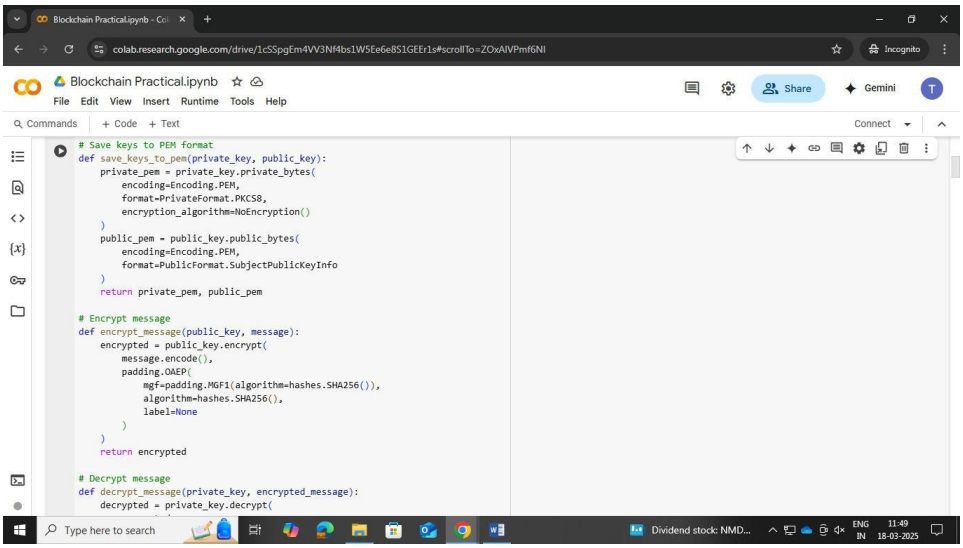
Code:

```
! 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
    )
```

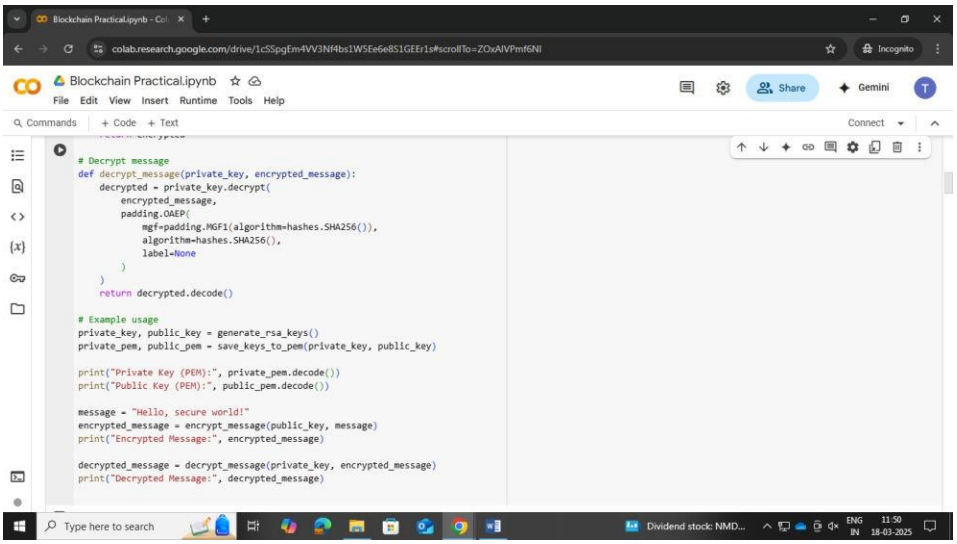
```
    return private_pem, public_pem

# 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)
```



```
! 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()
    )
```

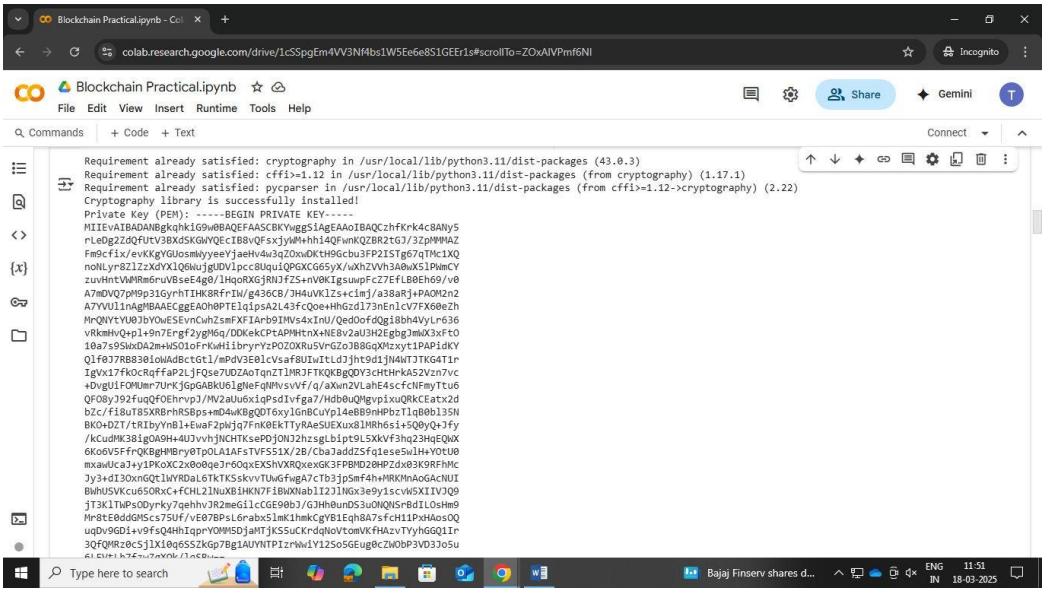


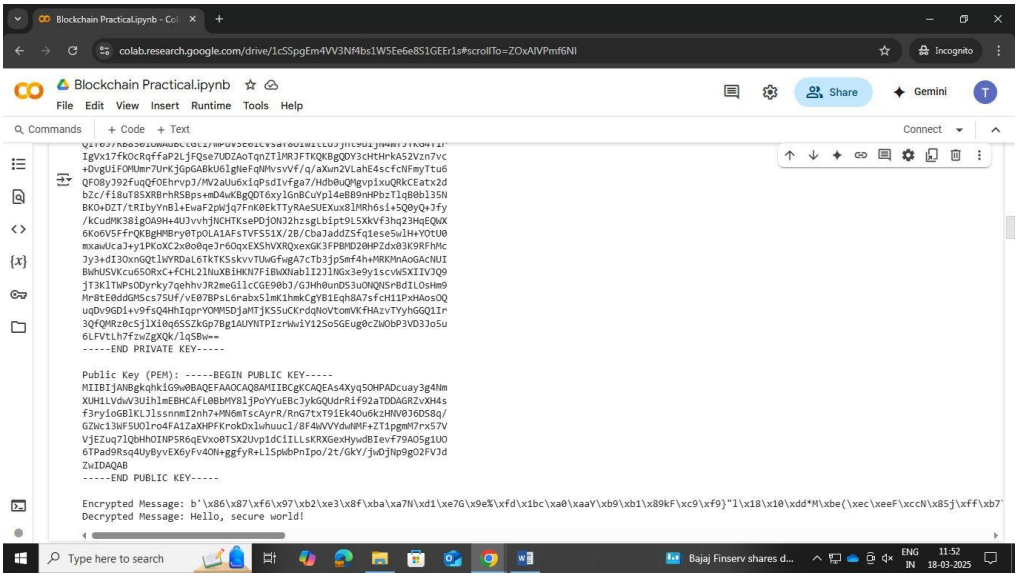
```
# Save keys to PEM format
def save_keys_to_pem(private_key, public_key):
    private_pem = private_key.private_bytes(
        encoding=Encoding.PEM,
        format=PrivateFormat.PKCS8,
        encryption_algorithm=NoEncryption()
    )
    public_pem = public_key.public_bytes(
        encoding=Encoding.PEM,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



```
# 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)
```

Output:


Blockchain Practical.ipynb - Col
colab.research.google.com/drive/1cSSpgEm4VV3Nf4bs1W5Ee6e8S1GEEr1s#scrollTo=ZOxAlVPmf6NI
Incognito
Blockchain Practical.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Connect
Requirement already satisfied: cryptography in /usr/local/lib/python3.11/dist-packages (43.0.3)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography) (1.17.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography) (2.22)
Cryptography library is successfully installed!
Private Key (PEM): -----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQCzhfKrk4c8ANy5
rLeDg2ZdQfUtV3BXdSKGWYQEcIB8vQFsxjyWM+hhi4QFwnKQZBR2tGJ/3ZpMMAZ
Fm9cfix/evKKgYGUosmkyyeeYjaeHv4w3qZOxwDKtH9Gcbu3FP2ISTg67qTMc1XQ
noNLyr8Zl2zXdYXlQ6WujgUDVlpcc8UquiQPGXCG65yX/wXhZVVh3A0wX5lPWmCY
zuvHntVWMRm6ruVBseE4g0/lHqoRXGjRNJfZS+nV0KIgsuwpFcZ7EfLB0Eh69/v0
A7mDVQ7pM9p31GyrhTIHK8RfrIW/g436CB/JH4uVKlZs+cimj/a38aRj+PAOM2n2
A7YVUl1nAgMBAAECggEAOh0PTElq1psA2L43fcQoe+HhGzdl73nEn1cV7FX60eZh
MrQNYtYU0JbYOwESEvnCwhZsmFXFIArb9IMVs4xInU/QedOofdQgi8bh4VyLr636
vRkmHvQ+pl+9n7Ergf2ygM6q/DDKekCPtAPMHtnX+NE8v2aU3H2EgbgJmWX3xFtO
10a7s9SWxDA2m+WSO1oFrKwH1ibryrYzPOZOXRu5VrGZoJB8GqXMzxyt1PAPidKY
Qlf0J7RB830ioWAdBctGtl/mPdV3E0lcVsaf8UIwItLdJjht9d1jN4WTJTKG4T1r
IgVx17fkOcRqffaP2LjFQse7UDZAoTqnZTlMRJFTKQKBgQDY3cHtHrkA52Vzn7vc
+DvgUiFOMUmr7UrKjGpGABkU6lgNeFqNMvsvVf/q/aXwn2VLahE4scfcNFmyTtu6
QFO8yJ92fuqQfOEhrvpJ/MV2aUu6x1qPsdIvfga7/Hdb0uQMgvpixuQRkCEatx2d
bZc/fi8uT85XRBrhRSBps+mD4wKBgQDT6xylGnBCuYp14eBB9nHPbzTlqB0bl35N
BKO+DZT/tRIbyYnBl+EwaF2pWjq7FnK0EkTTyRAeSUEXux8lMRh6si+5Q0yQ+Jfy
/kCudMK38igOA9H+4UJvvhjNCHTKsePDjON2hzsgLbipt9L5XkVf3hq23HqEQkX
6Ko6V5FfrQKBgHMBry0TpOLA1AFsTVFS51X/2B/CbaJaddZSfq1ese5wlH+YOtU0
mxawUcaJ+y1PKoXC2x0o0qeJr6OqxEXShVXRQxexGK3FPBMD28HPZdx03K9RFhMc
Jy3+dI3OxnGQtlWYRDaL6TkTKSskvvTUwGfwgA7cTb3jpSmf4h+MRKMnAoGAcNUI
BWhUSVKcu65ORxC+fCHL2lNuXBiHKN7FiBWXNablI2JlNGx3e9y1scvW5XIIVJQ9
jT3KlTWPsODyrky7qehhvJR2meGilcCGE90bJ/GJHh0unDS3uONQNSrBdILOsHm9
Mr8tE0ddGMScs75Uf/vE07BPsL6rabx5lmK1hmkCgYB1Eqh8A7sfcH11PxHAosOQ
uqDv9GDi+v9fsQ4HhIqprYOMM5DjaMTjKS5uCKrdqNoVtomVKfHAzvTYyhGGQ1Ir
3QfQMRz0cSjlXi0q6SSZkGp7Bg1AUYNTPIzrWwiY12SoSGEug0cZWObP3VD3Jo5u
6LFVtLh7fzwZgXQk/lqSBw==
-----END PRIVATE KEY-----
Public Key (PEM): -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs4Xyq5OHPADcuay3g4Nm
XUH1LVdwV3UihlmEBHCAfL0BbMY8ljPoYYuEBcJykGQUdrRif92aTDDAGRZvXH4s
f3ryloGBlKLJlssnnmI2nh7+MN6mTscAyrR/RnG7txT9lEk4OubkzHNV036DS8q/
GZWc13WF5UO1ro4FA1ZaXHPFKrokDxlwhuucl/8F4WVYdwNMF+ZT1pgmM7rx57V
VjEZuq7lQbHhOINPSR6qEVxo0TSX2Uvp1dCiILLsKRXGexHywdBIevf79AO5g1UO
6TPad9Rsq4UyByvEX6yFv4ON+ggfyR+LlSpWbPnIpo/2t/GkY/jwDjNp9gO2FVJd
ZwIDAQAB
-----END PUBLIC KEY-----
Encrypted Message: b'\x86\x87\xf6\x97\xb2\xe3\x8f\xba\xa7N\xd1\xe7G\x9e%\xfd\x1bc\xa0\xaaY\xb9\xb1\x89kF\xc9\xf9}"1\x18\x10\xdd*M\xbe(\xec\xeeF\xccN\x85j\xff\xb7
Decrypted Message: Hello, secure world!
Type here to search
Bajaj Finserv shares d...
ENG IN
11:51
18-03-2025


Blockchain Practical.ipynb - Col
colab.research.google.com/drive/1cSSpgEm4VV3Nf4bs1W5Ee6e8S1GEEr1s#scrollTo=ZOxAlVPmf6NI
Incognito
Blockchain Practical.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Connect
IgVx17fkOcRqffaP2LjFQse7UDZAoTqnZTlMRJFTKQKBgQDY3cHtHrkA52Vzn7vc
+DvgUiFOMUmr7UrKjGpGABkU6lgNeFqNMvsvVf/q/aXwn2VLahE4scfcNFmyTtu6
QFO8yJ92fuqQfOEhrvpJ/MV2aUu6x1qPsdIvfga7/Hdb0uQMgvpixuQRkCEatx2d
bZc/fi8uT85XRBrhRSBps+mD4wKBgQDT6xylGnBCuYp14eBB9nHPbzTlqB0bl35N
BKO+DZT/tRIbyYnBl+EwaF2pWjq7FnK0EkTTyRAeSUEXux8lMRh6si+5Q0yQ+Jfy
/kCudMK38igOA9H+4UJvvhjNCHTKsePDjON2hzsgLbipt9L5XkVf3hq23HqEQkX
6Ko6V5FfrQKBgHMBry0TpOLA1AFsTVFS51X/2B/CbaJaddZSfq1ese5wlH+YOtU0
mxawUcaJ+y1PKoXC2x0o0qeJr6OqxEXShVXRQxexGK3FPBMD28HPZdx03K9RFhMc
Jy3+dI3OxnGQtlWYRDaL6TkTKSskvvTUwGfwgA7cTb3jpSmf4h+MRKMnAoGAcNUI
BWhUSVKcu65ORxC+fCHL2lNuXBiHKN7FiBWXNablI2JlNGx3e9y1scvW5XIIVJQ9
jT3KlTWPsODyrky7qehhvJR2meGilcCGE90bJ/GJHh0unDS3uONQNSrBdILOsHm9
Mr8tE0ddGMScs75Uf/vE07BPsL6rabx5lmK1hmkCgYB1Eqh8A7sfcH11PxHAosOQ
uqDv9GDi+v9fsQ4HhIqprYOMM5DjaMTjKS5uCKrdqNoVtomVKfHAzvTYyhGGQ1Ir
3QfQMRz0cSjlXi0q6SSZkGp7Bg1AUYNTPIzrWwiY12SoSGEug0cZWObP3VD3Jo5u
6LFVtLh7fzwZgXQk/lqSBw==
-----END PRIVATE KEY-----
Public Key (PEM): -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs4Xyq5OHPADcuay3g4Nm
XUH1LVdwV3UihlmEBHCAfL0BbMY8ljPoYYuEBcJykGQUdrRif92aTDDAGRZvXH4s
f3ryloGBlKLJlssnnmI2nh7+MN6mTscAyrR/RnG7txT9lEk4OubkzHNV036DS8q/
GZWc13WF5UO1ro4FA1ZaXHPFKrokDxlwhuucl/8F4WVYdwNMF+ZT1pgmM7rx57V
VjEZuq7lQbHhOINPSR6qEVxo0TSX2Uvp1dCiILLsKRXGexHywdBIevf79AO5g1UO
6TPad9Rsq4UyByvEX6yFv4ON+ggfyR+LlSpWbPnIpo/2t/GkY/jwDjNp9gO2FVJd
ZwIDAQAB
-----END PUBLIC KEY-----
Encrypted Message: b'\x86\x87\xf6\x97\xb2\xe3\x8f\xba\xa7N\xd1\xe7G\x9e%\xfd\x1bc\xa0\xaaY\xb9\xb1\x89kF\xc9\xf9}"1\x18\x10\xdd*M\xbe(\xec\xeeF\xccN\x85j\xff\xb7
Decrypted Message: Hello, secure world!
Type here to search
Bajaj Finserv shares d...
ENG IN
11:52
18-03-2025

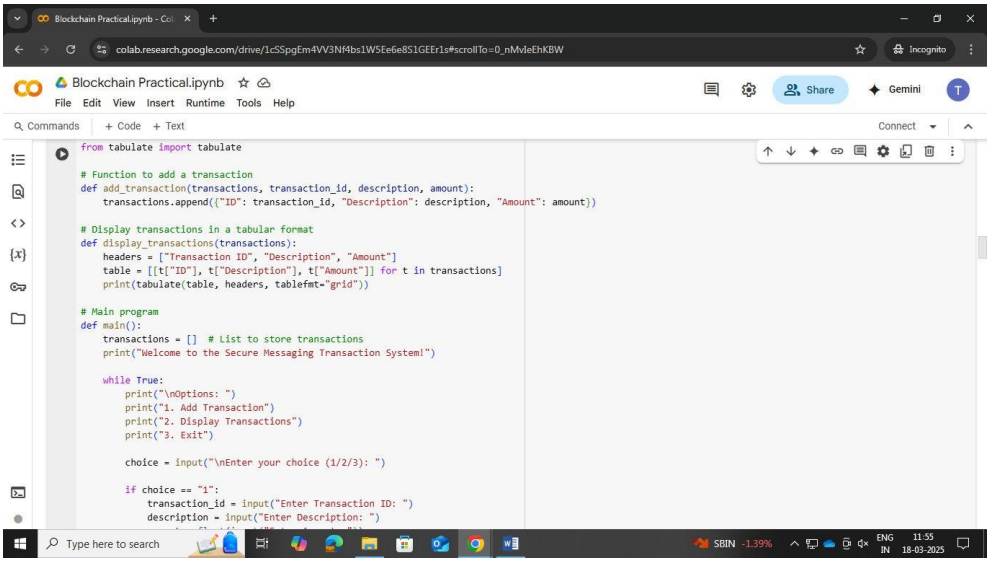
PRACTICAL 1 B.

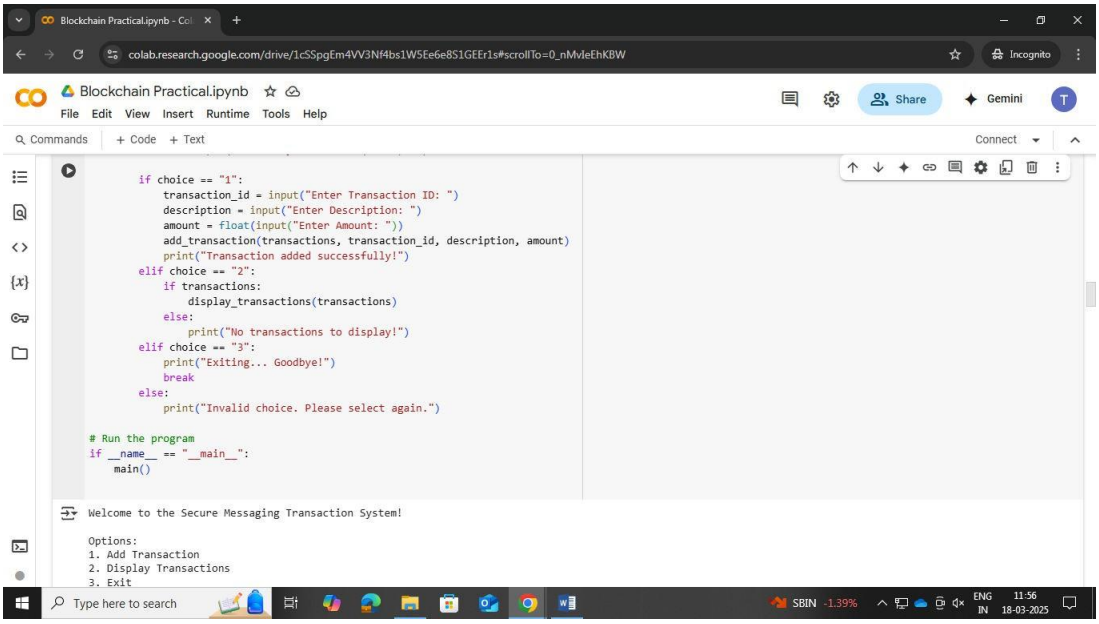
Aim: Allow users to create multiple transactions and display them in an organised format.

Code:

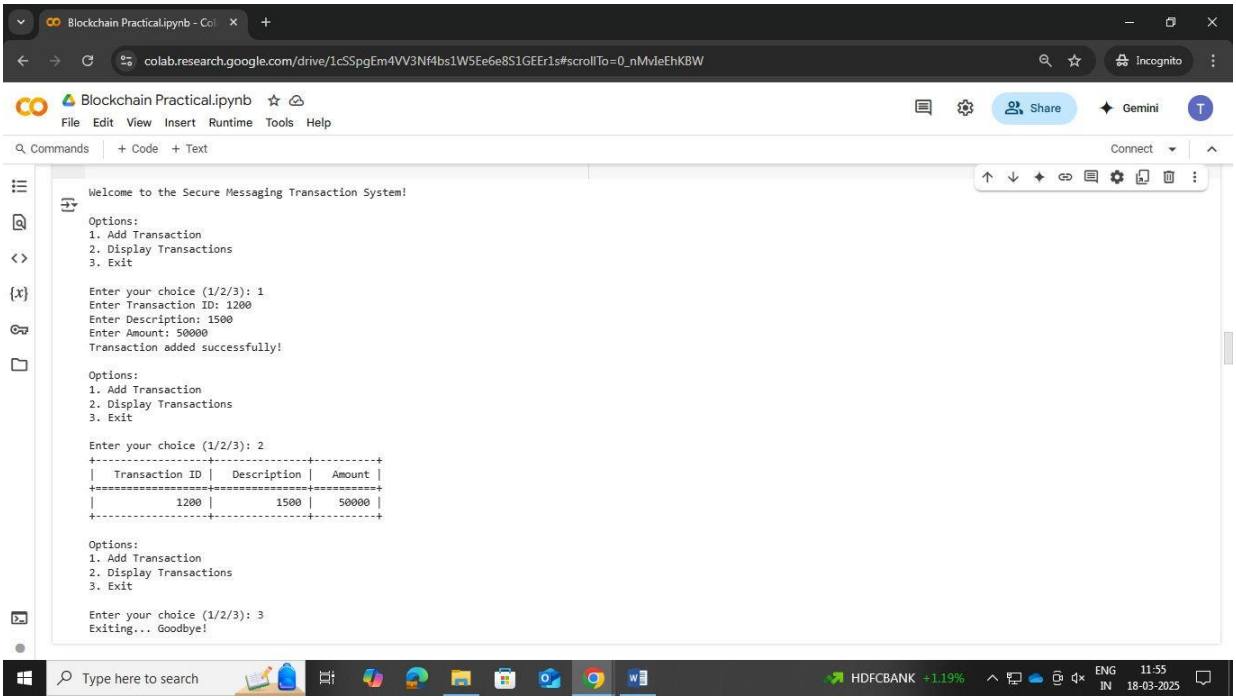
```
from tabulate import tabulate

# 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()
```





Output:


Blockchain Practical.ipynb - Col
colab.research.google.com/drive/1cSSpgEm4VV3Nf4bs1W5Ee6e8S1GEEr1s#scrollTo=0_nMvIeEhKBW
Incognito
Blockchain Practical.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Connect
Welcome to the Secure Messaging Transaction System!
Options:
1. Add Transaction
2. Display Transactions
3. Exit
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!
Type here to search
HDFCBANK +1.19%
ENG
IN
11:55
18-03-2025

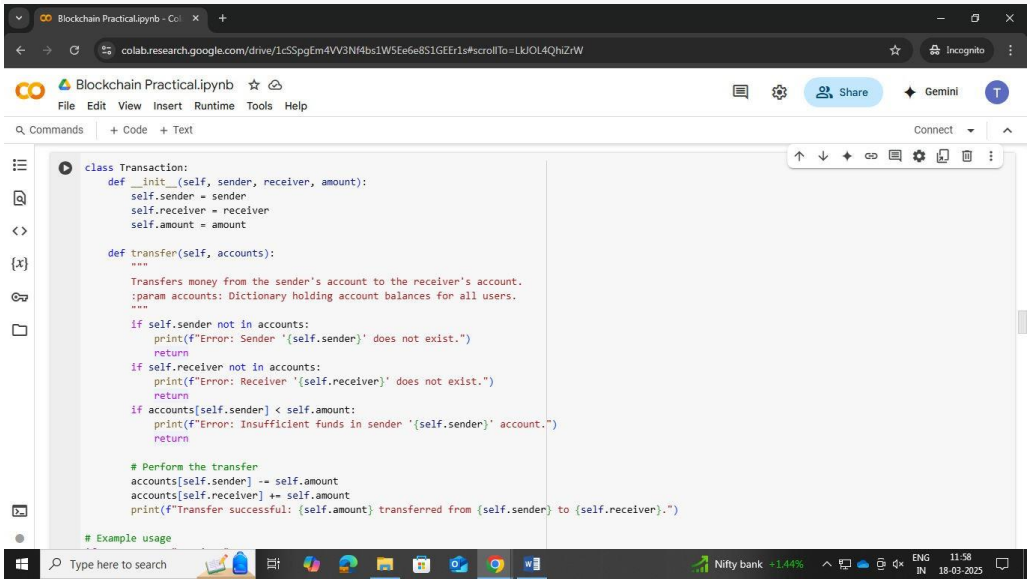
PRACTICAL 1 C.

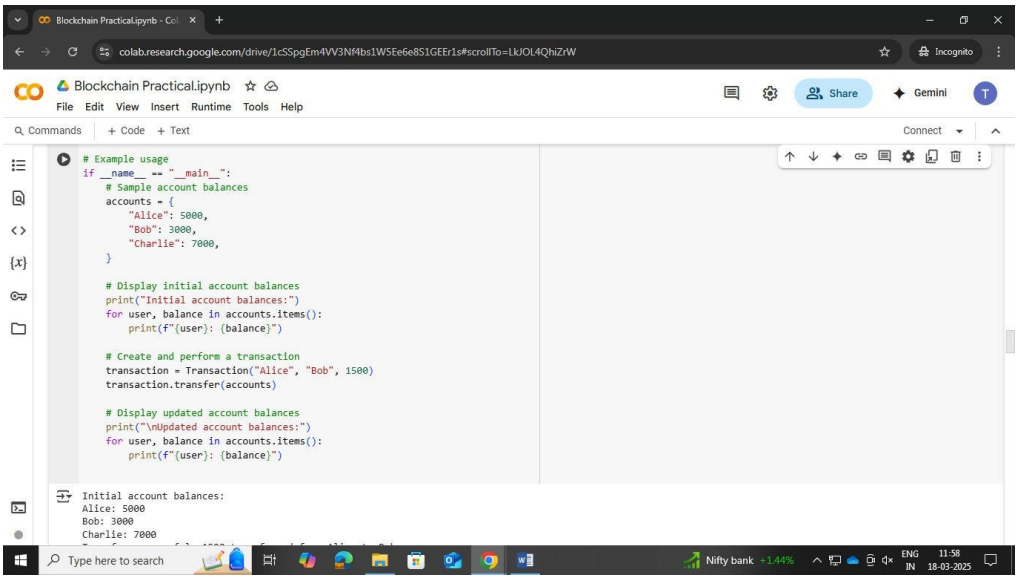
Aim: Create a Python class named Transaction with attributes for sender, receiver, and amount. Implement a method within the class to transfer money from the sender's account to the receiver's account.

Code:

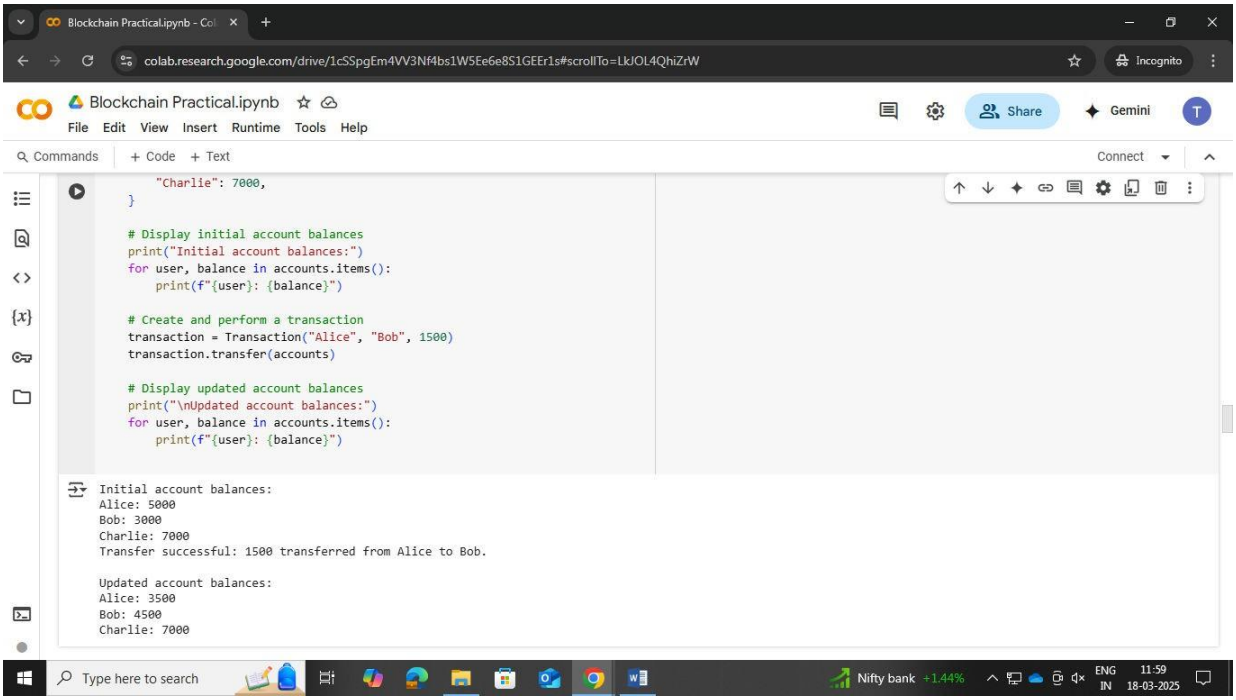
```
class Transaction:
    def __init__(self, sender, receiver, amount):
        self.sender = sender
        self.receiver = receiver
        self.amount = amount
    def transfer(self, accounts):
        """
        Transfers money from the sender's account to the receiver's
account.
        :param 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}")
```





Output:


Blockchain Practical.ipynb - Col
colab.research.google.com/drive/1cSSpgEm4VV3Nf4bs1W5Ee6e8S1GEEr1s#scrollTo=LkJOL4QhiZrW
Incognito
Blockchain Practical.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Connect
"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
Type here to search
Nifty bank +1.44%
ENG
IN
11:59
18-03-2025

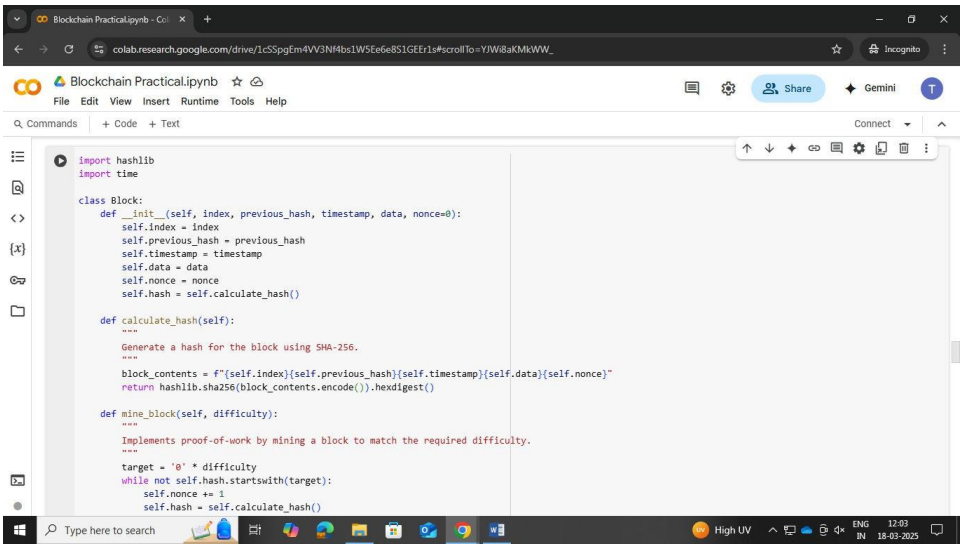
PRACTICAL 1 D.

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

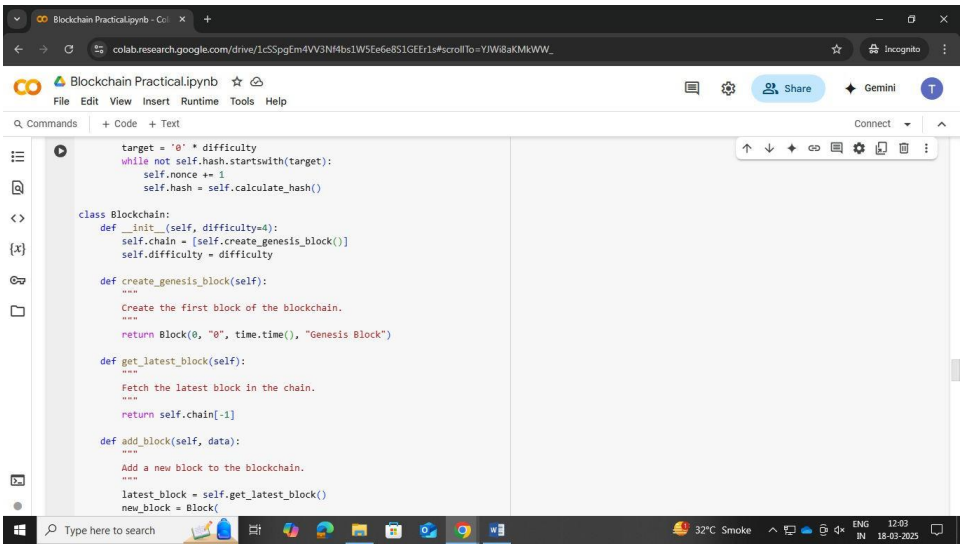
Code:

```
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")
```

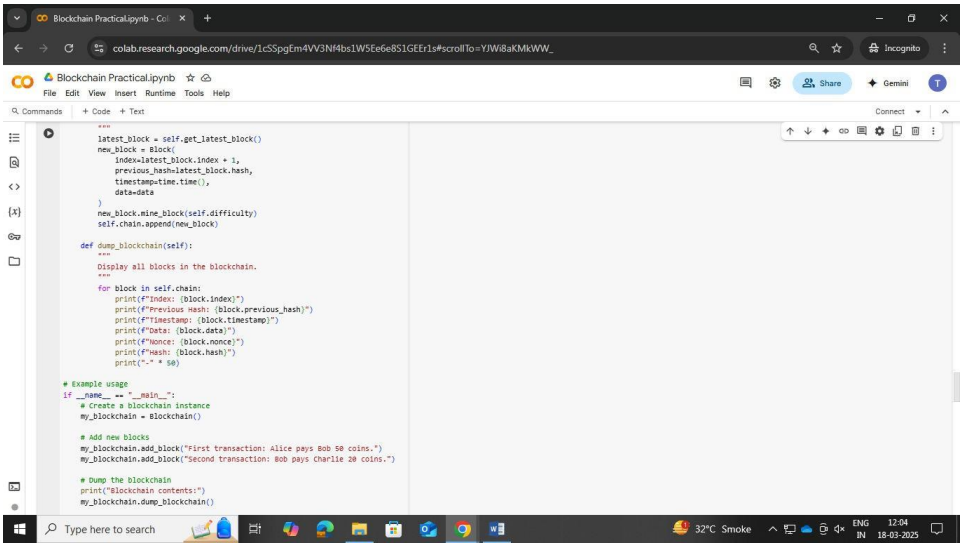
```
    def get_latest_block(self):
        """
        Fetch the latest block in the chain.
        """
        return self.chain[-1]

    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()
```



```
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()
```

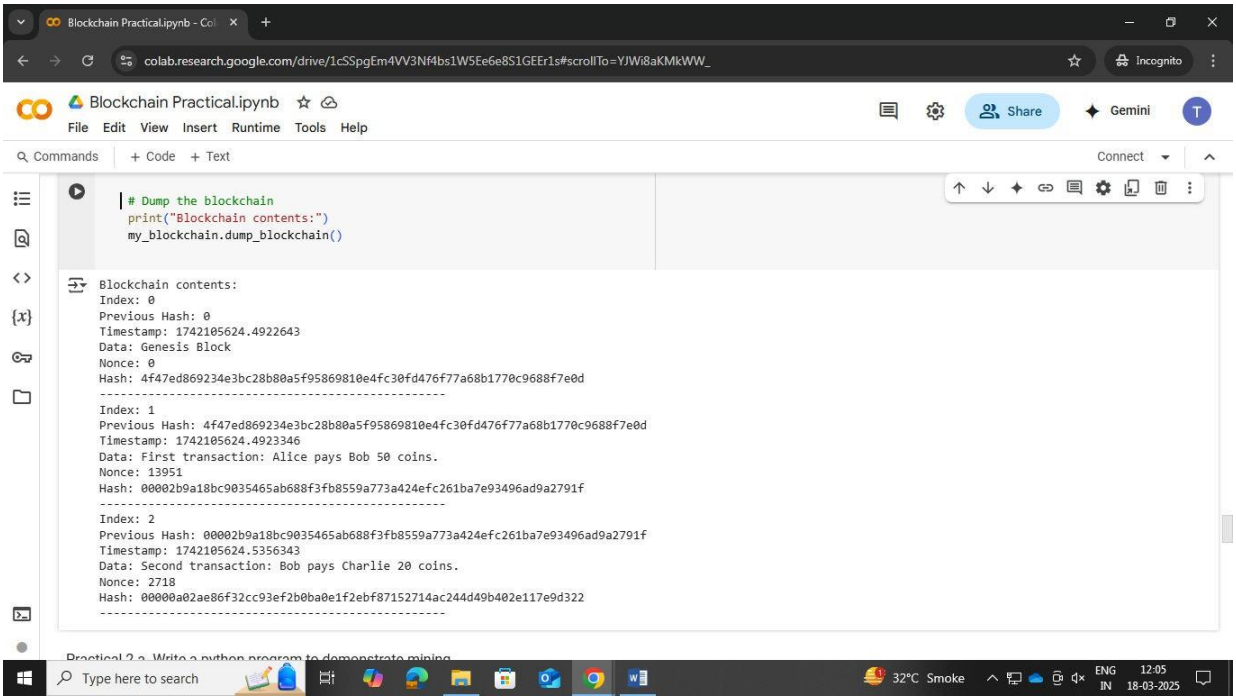


```
        target = '0' * difficulty
        while not self.hash.startswith(target):
            self.nonce += 1
            self.hash = self.calculate_hash()

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()
```

Output:

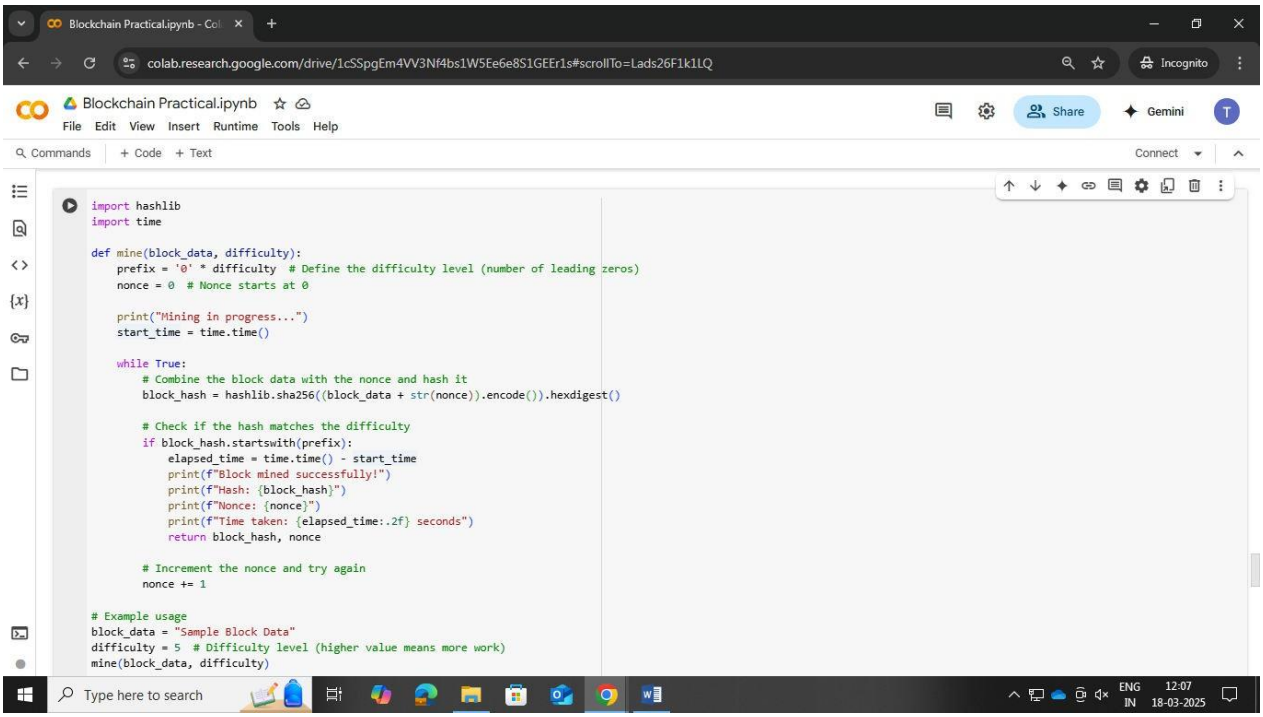

Blockchain Practical.ipynb - Col
colab.research.google.com/drive/1cSSpgEm4VV3Nf4bs1W5Ee6e8S1GEEr1s#scrollTo=YJWi8aKMkWW_
Incognito
Blockchain Practical.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Connect
# Dump the blockchain
print("Blockchain contents:")
my_blockchain.dump_blockchain()
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
Type here to search
32°C Smoke
ENG
IN
12:05
18-03-2025

PRACTICAL 2 A.

Aim: Write a python program to demonstrate mining.

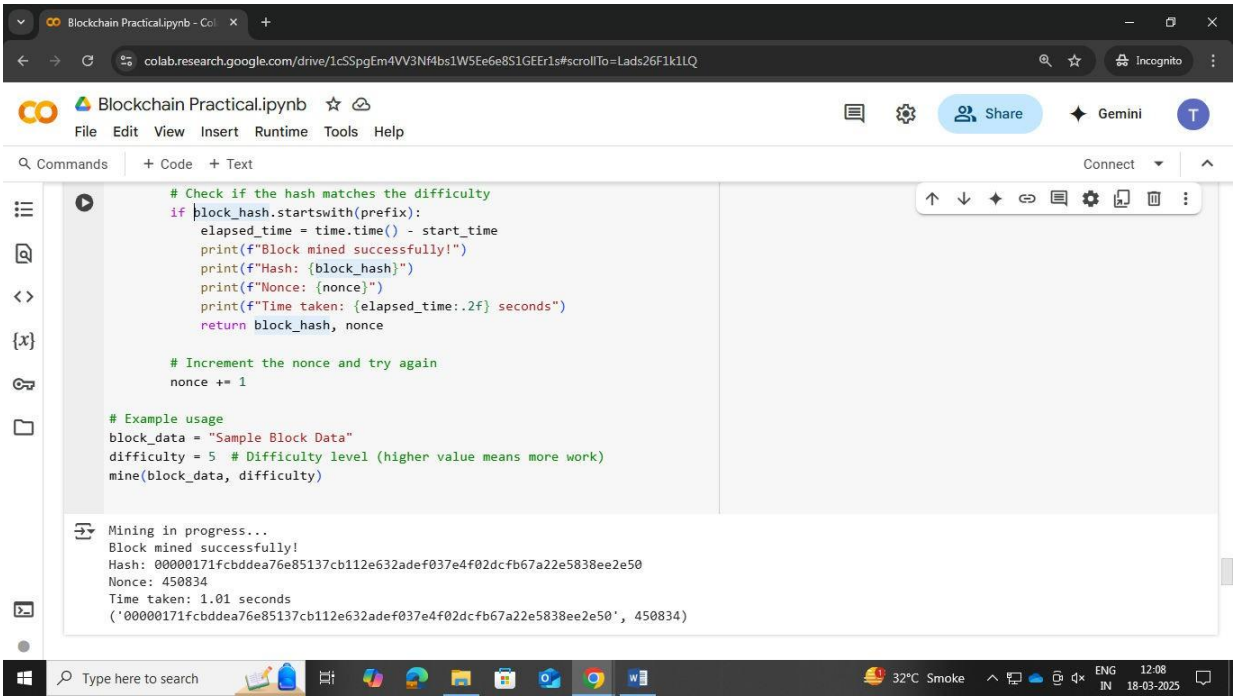
Code:

```
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)
```



```
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:



```
        # 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: 00000171fcbddea76e85137cb112e632adef037e4f02dcfb67a22e5838ee2e50
Nonce: 450834
Time taken: 1.01 seconds
('00000171fcbddea76e85137cb112e632adef037e4f02dcfb67a22e5838ee2e50', 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

Blockchain Info: {'chain': 'main', 'blocks': 768914, 'headers': 768914, 'bestblockhash': '00000000000000000b1a29528e897e63fba5952f17f3a8f6b0c60ad826c71b', 'difficulty': 218938744019.67, 'mediantime': 1628537400, 'verificationprogress': 0.999999, 'initialblockdownload': False, 'chainwork': '0000000000000000000000000000000000000000000000000017ea7', 'size_on_disk': 35101620386, 'pruned': False, 'softforks': [], 'bip9_softforks': {}}

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": "0x0000000000000000000000000000000000000000",
  "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()
```

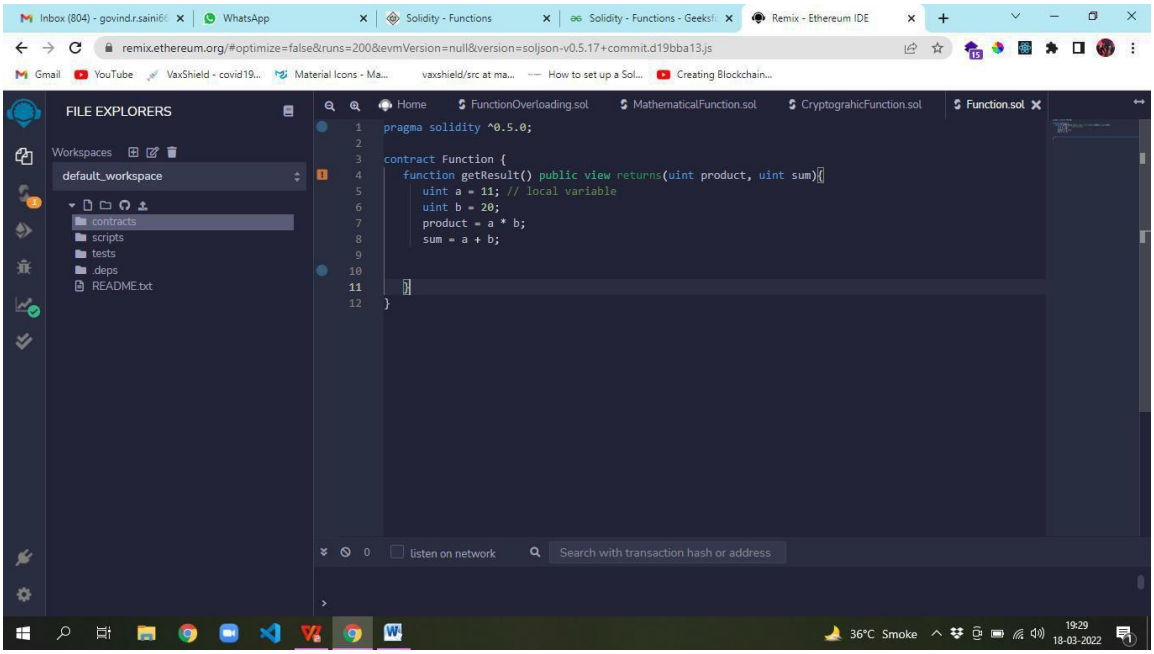
PRACTICAL 3 A.

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

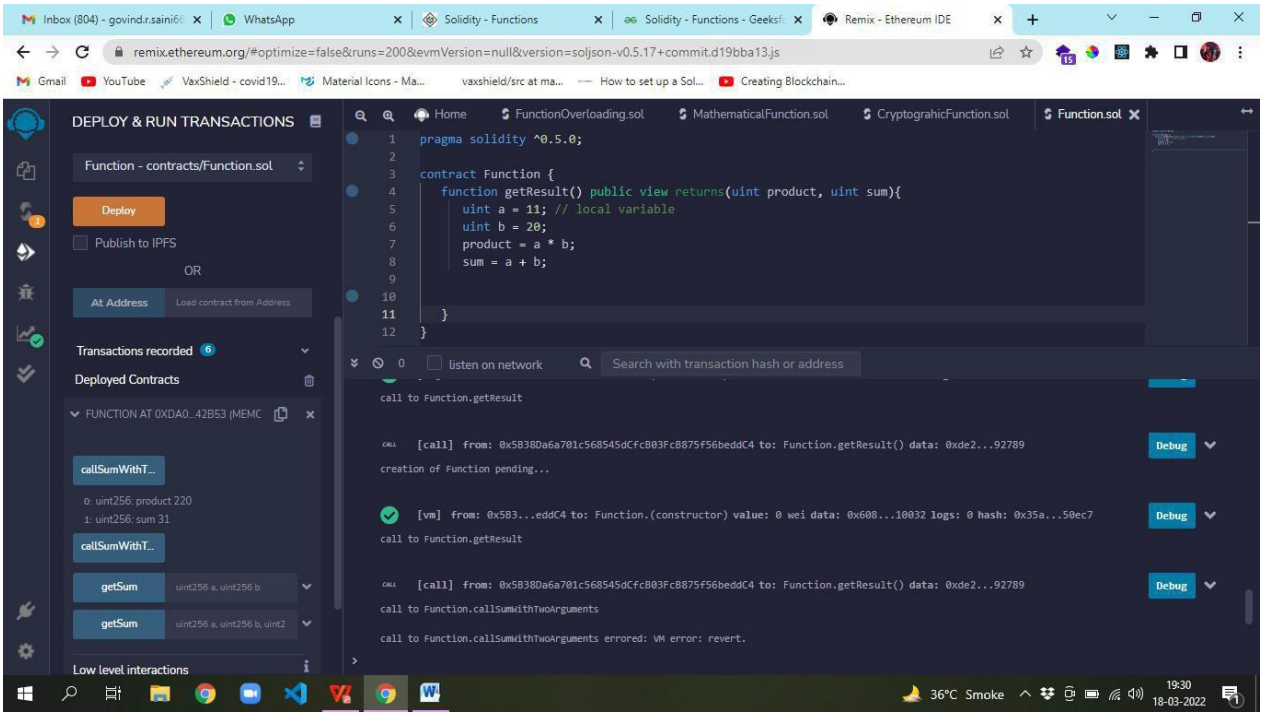
Code:

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;

}

}
```



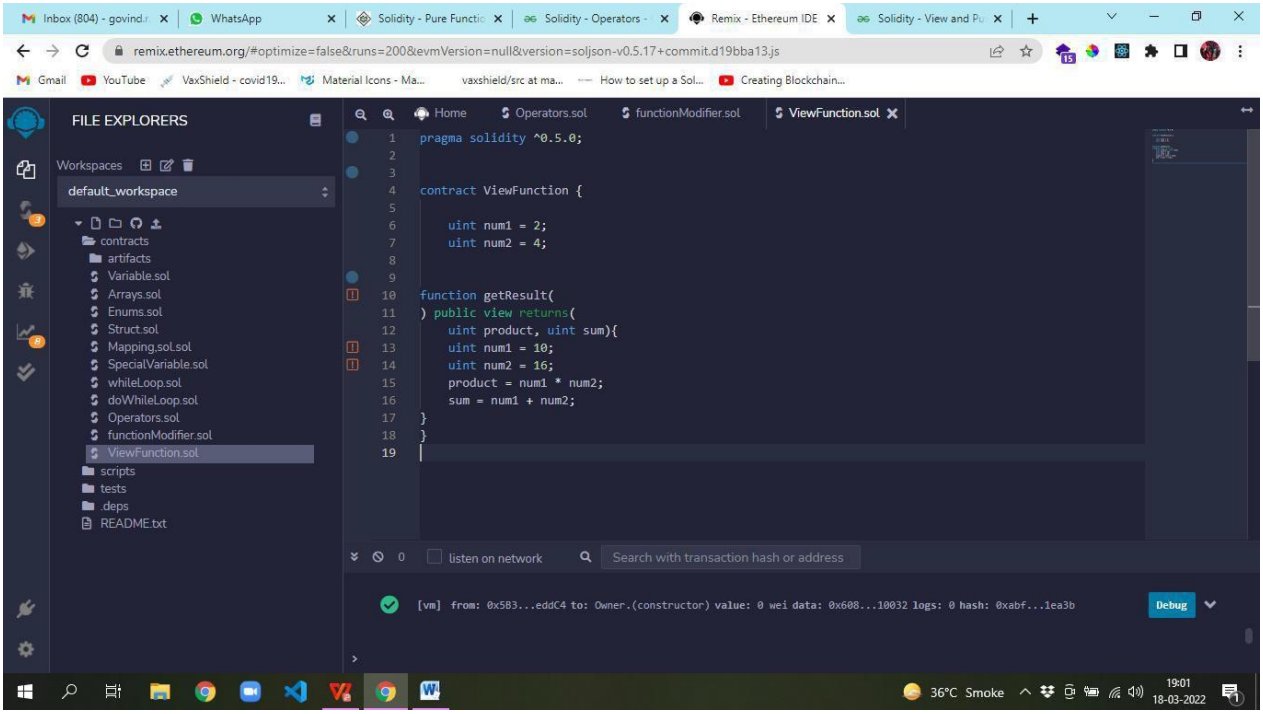
Output:



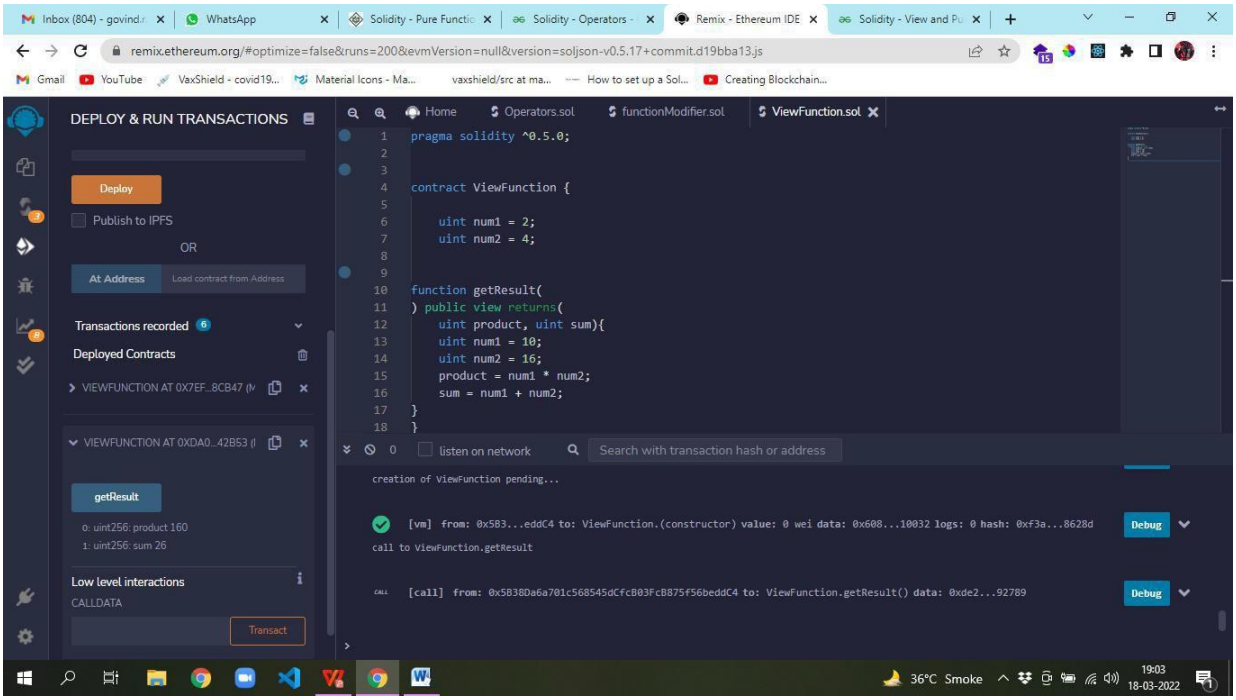
View Function

```
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;

}

}
```

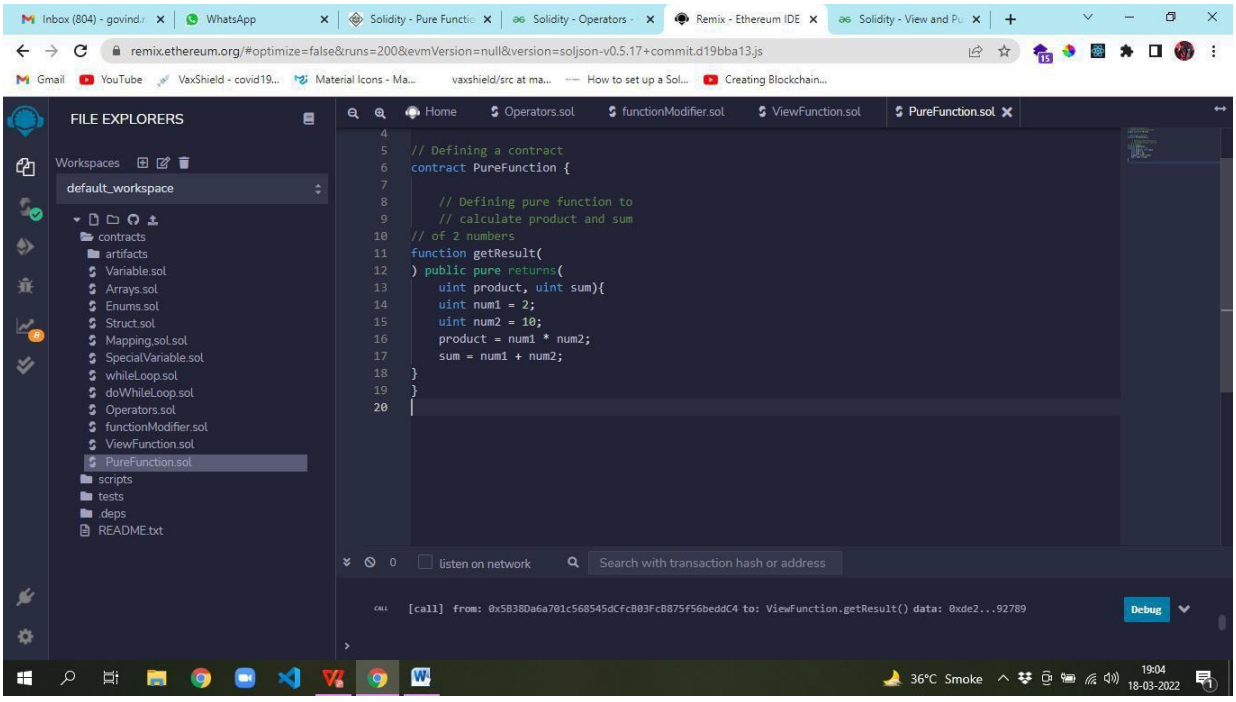


Output:

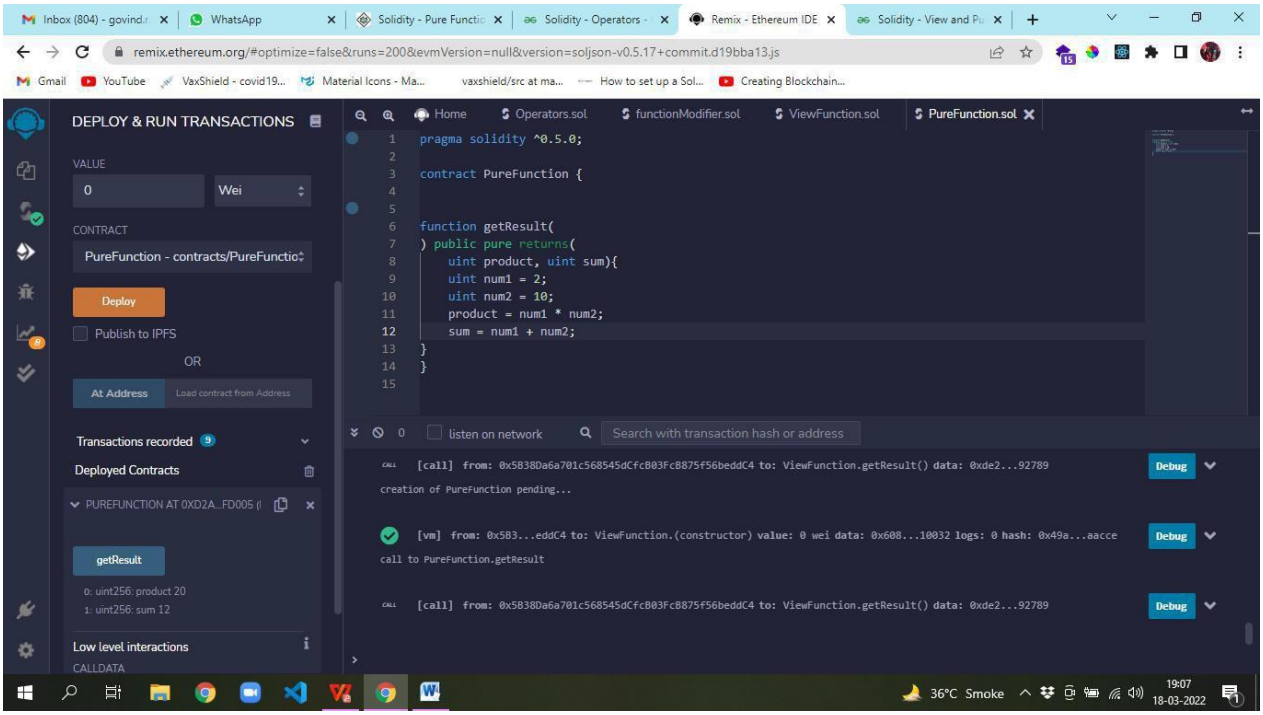


Pure Function

```
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;

}

}
```



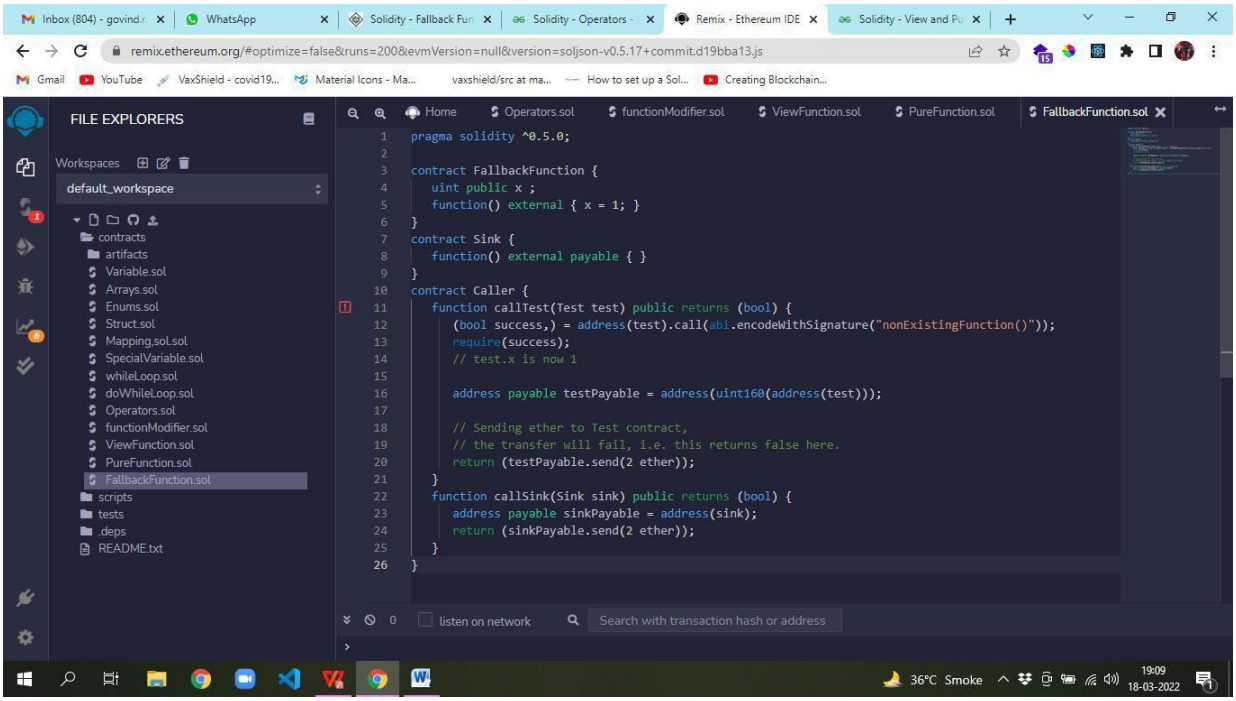
Output:



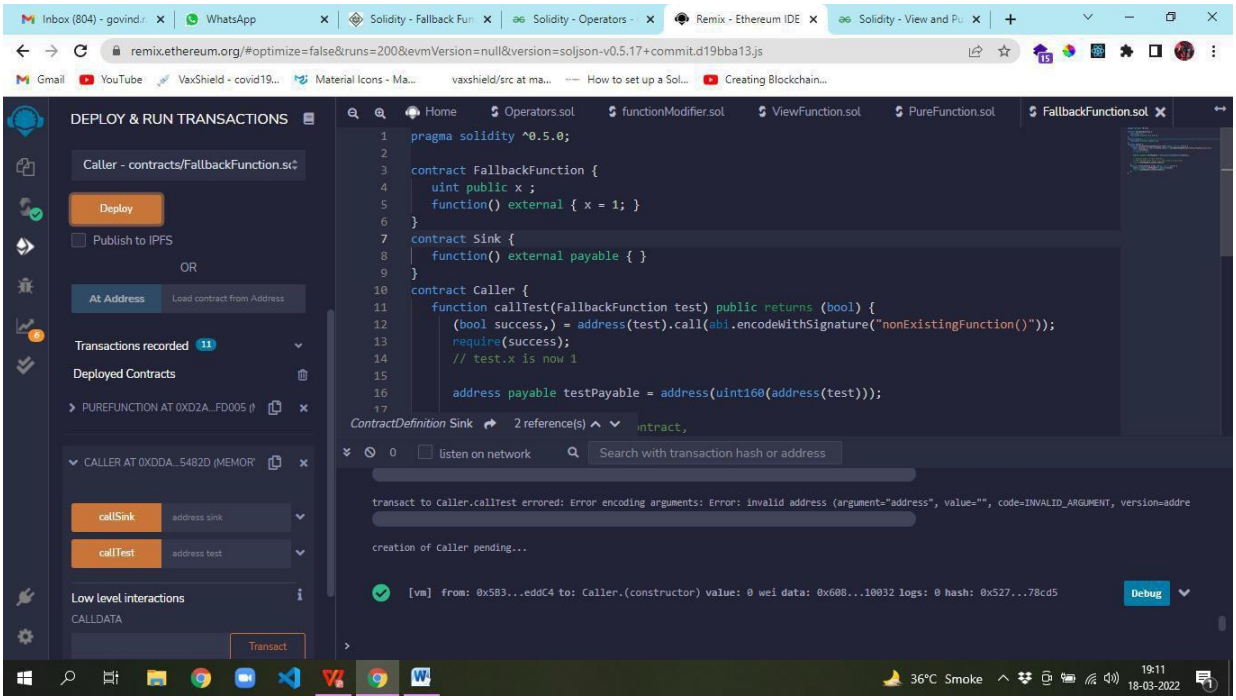
Fallback Function

```
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));
}
}
```



Output:


DEPLOY & RUN TRANSACTIONS
Caller - contracts/FallbackFunction.sol
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 11
Deployed Contracts
callSink
callTest
Low level interactions
CALLDATA
Transact
Home
Operators.sol
functionModifier.sol
ViewFunction.sol
PureFunction.sol
FallbackFunction.sol
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)));
transact to Caller.callTest errored: Error encoding arguments: Error: invalid address (argument="address", value="", code=INVALID_ARGUMENT, version=addre
creation of Caller pending...
[vm] from: 0x583...eddC4 to: Caller.(constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0x527...78cd5
Debug
Search with transaction hash or address
listen on network
36°C Smoke
19:11
18-03-2022

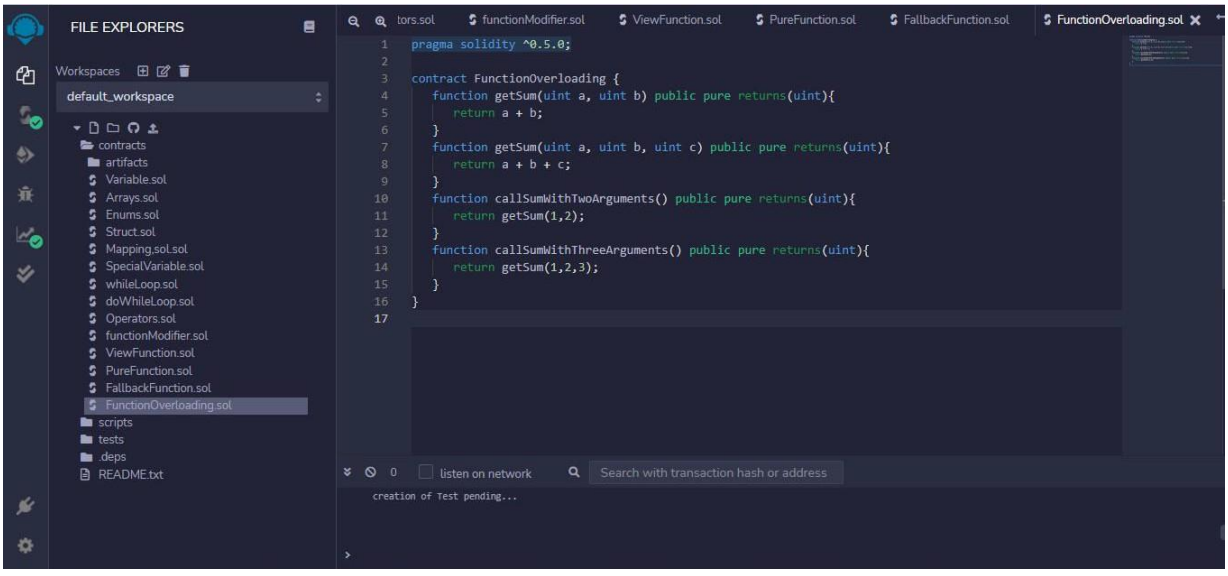
PRACTICAL 3 B.

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

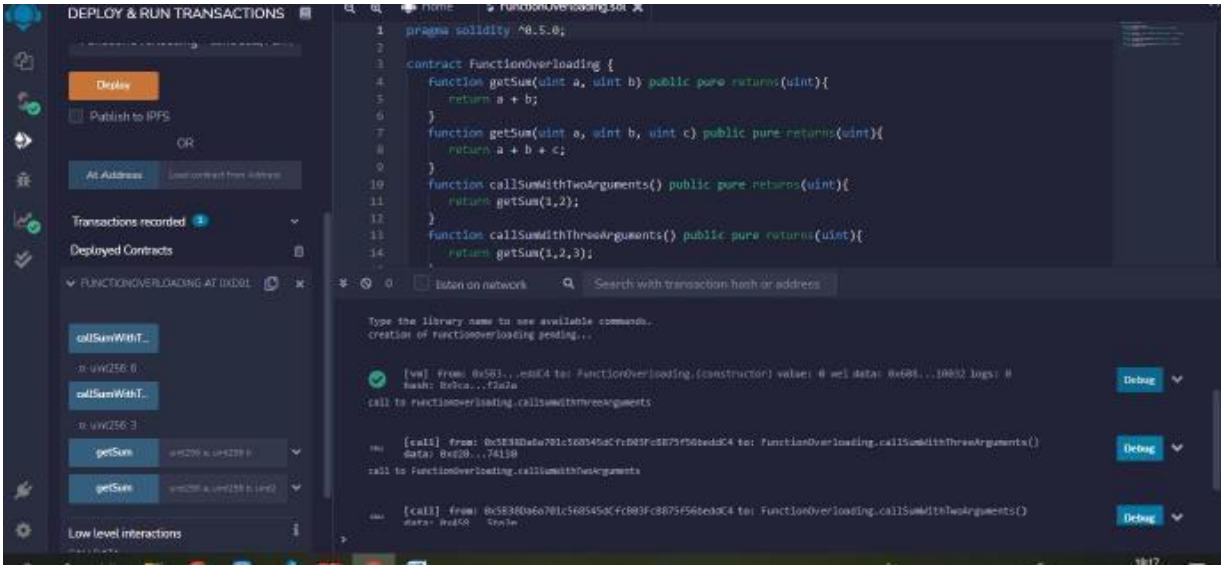
Code:

// 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;

   }
```

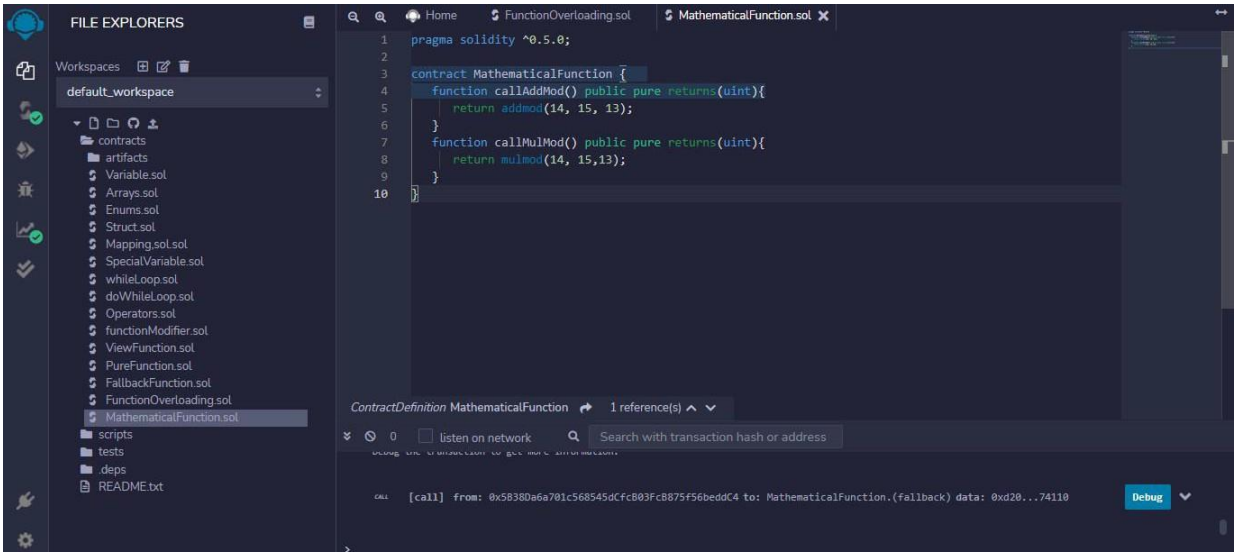


Output

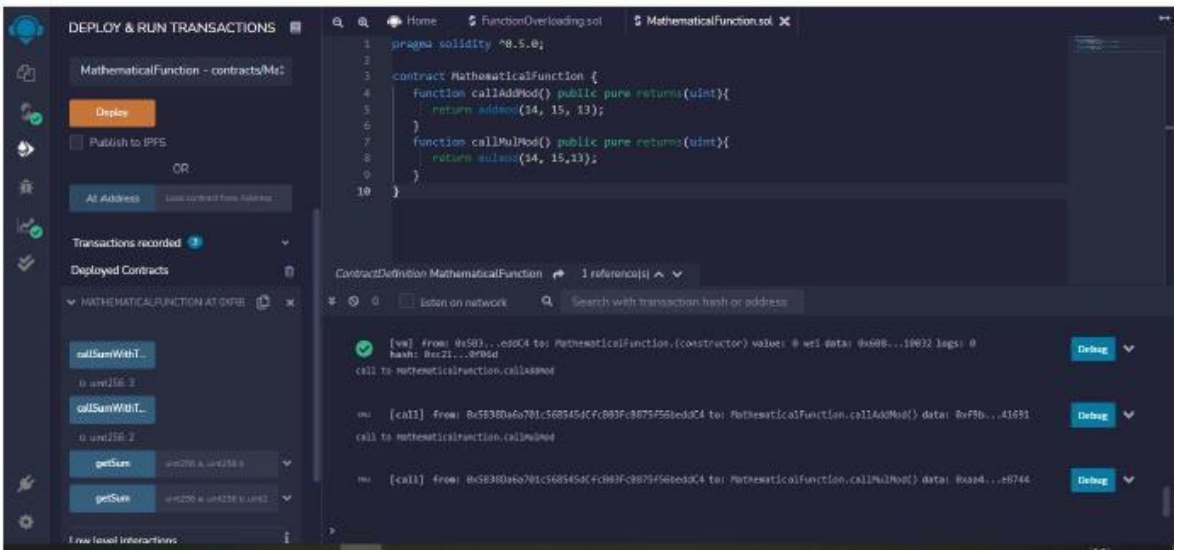


// Mathematical functions

```
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;
}
```

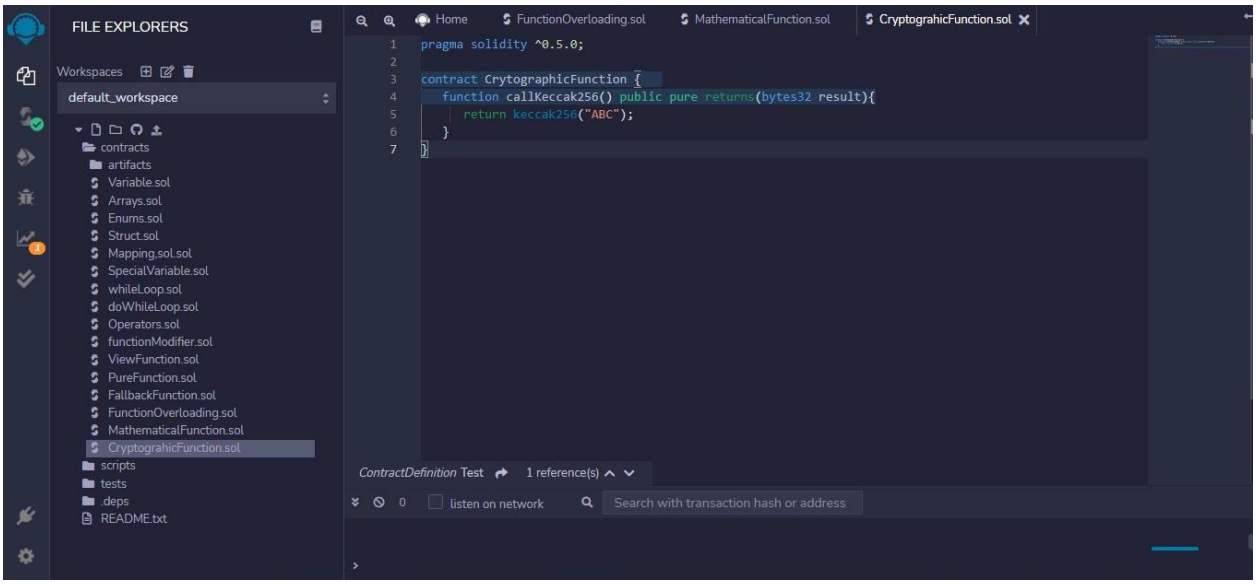


Output

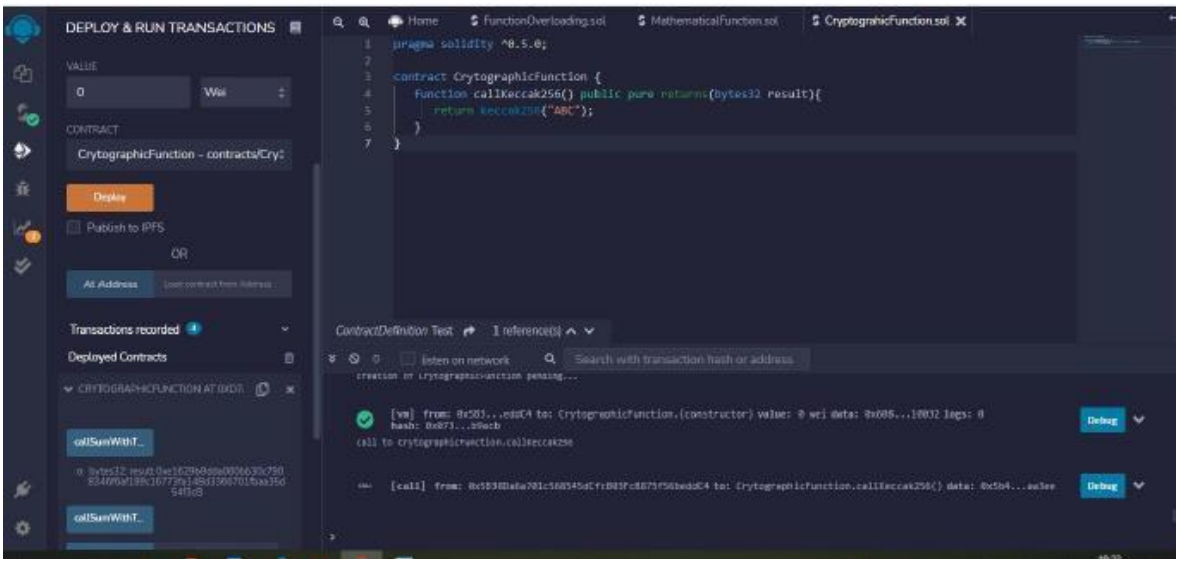


// Cryptographic functions

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



Output:


DEPLOY & RUN TRANSACTIONS
VALUE
0
Wei
CONTRACT
CrytographicFunction - contracts/Cry
Deploy
Publish to IPFS
OR
At Address
Transactions recorded
Deployed Contracts
callSumWithT...
callSumWithT...
Home
FunctionOverloading.sol
MathematicalFunction.sol
CryptographicFunction.sol
pragma solidity ^0.5.0;
contract CrytographicFunction {
function callKeccak256() public pure returns(bytes32 result){
return keccak256("ABC");
}
}
ContractDefinition Test
1 reference(s)
listen on network
Search with transaction hash or address
creation of CrytographicFunction pending...
[vm] from: 0x583...eddC4 to: CrytographicFunction.(constructor) value: 0 wei data: 0x608...10032 logs: 0
hash: 0x873...a9acb
call to CrytographicFunction.callKeccak256
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: CrytographicFunction.callKeccak256() data: 0x5b4...ea5ee
Debug
Debug

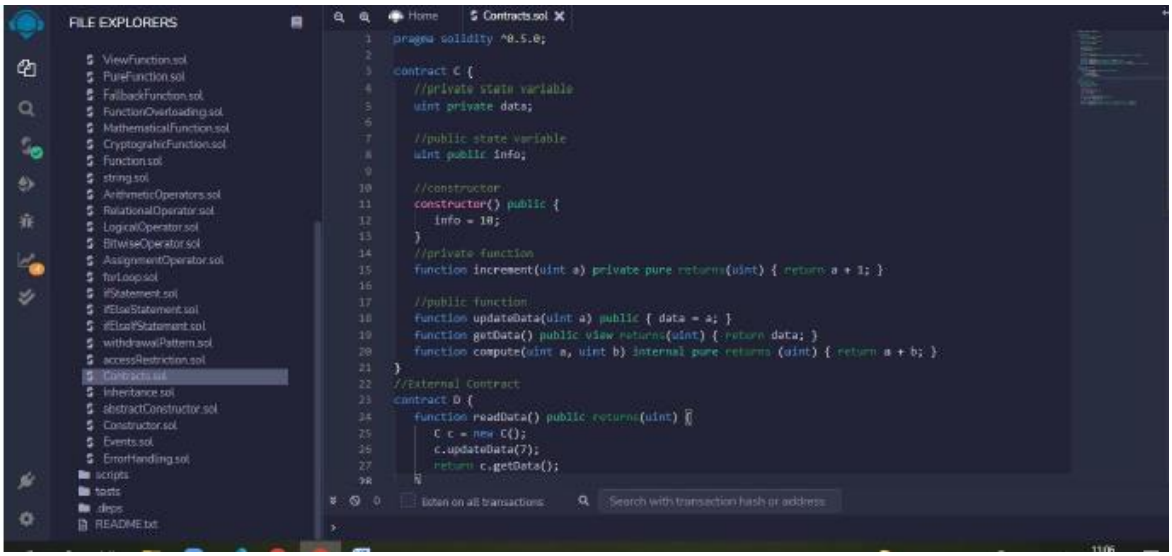
PRACTICAL 3 C.

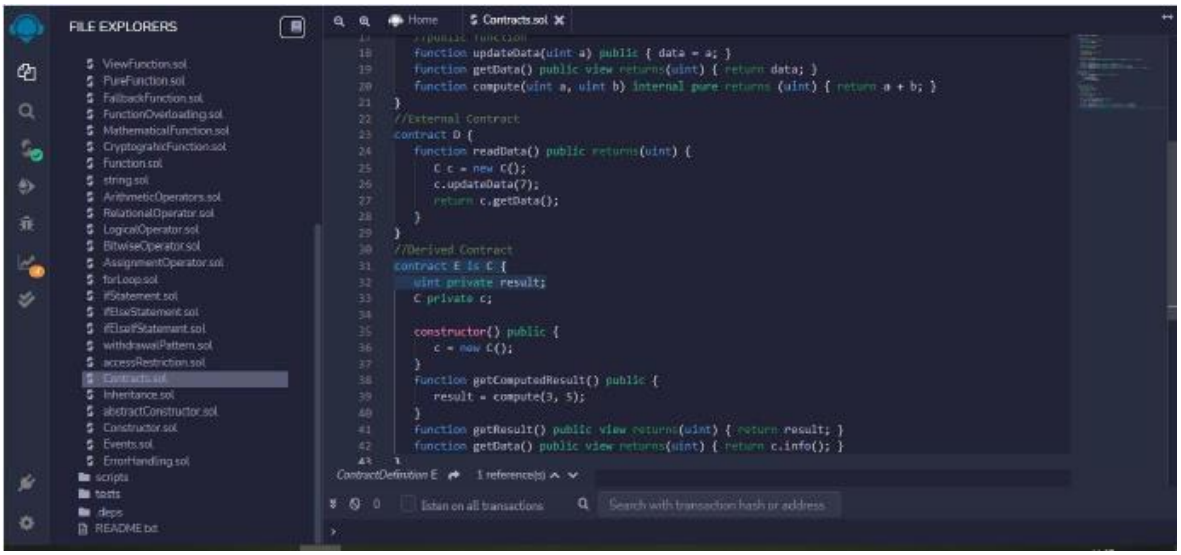
Aim: Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.

Contracts:

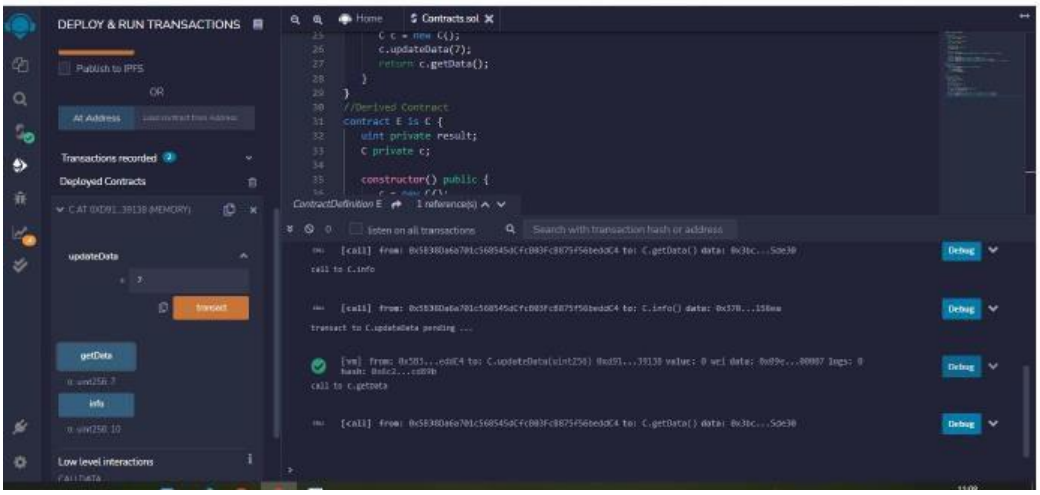
Code:

```
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;
    }
}
```


FILE EXPLORERS
Home
Contracts.sol
pragma solidity ^0.5.0;
contract C {
//private state variable
uint private data;
//public state variable
uint public info;
//constructor
constructor() public {
info = 10;
}
//private function
function increment(uint a) private pure returns(uint) { return a + 1; }
//public function
function updateData(uint a) public { data = a; }
function getData() public view returns(uint) { return data; }
function compute(uint a, uint b) internal pure returns (uint) { return a + b; }
}
//External Contract
contract D {
function readData() public returns(uint) {
C c = new C();
c.updateData(7);
return c.getData();
}
}


FILE EXPLORERS
Home
Contracts.sol
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; }
}
//External Contract
contract D {
function readData() public returns(uint) {
C c = new C();
c.updateData(7);
return c.getData();
}
}
//Derived Contract
contract E is C {
uint private result;
C private c;
constructor() public {
c = new C();
}
function getComputedResult() public {
result = compute(3, 5);
}
function getResult() public view returns(uint) { return result; }
function getData() public view returns(uint) { return c.info(); }
}
ContractDefinition E
1 reference(s)
Listen on all transactions
Search with transaction hash or address

Output:

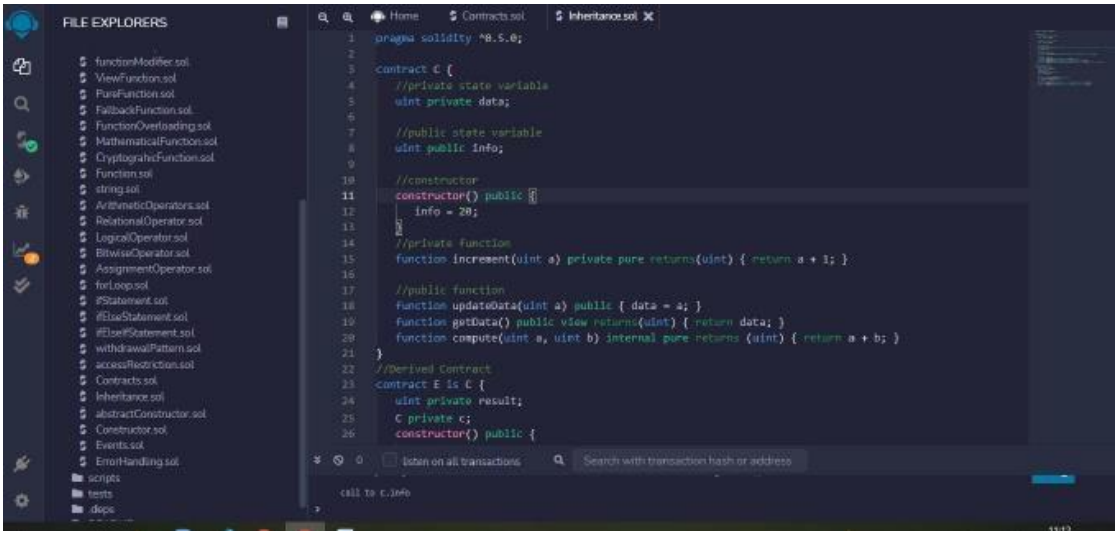

DEPLOY & RUN TRANSACTIONS
Publish to IPFS
OR
At Address
Transactions recorded
Deployed Contracts
updateData
transact
getData
info
Low level interactions
Home
Contracts.sol
c.updateData(7);
return c.getData();
}
}
//Derived Contract
contract E is C {
uint private result;
C private c;
constructor() public {
ContractDefinition E
1 reference(s)
Listen on all transactions
Search with transaction hash or address
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: C.getData() data: 0x3bc...50e30
call to C.info
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: C.info() data: 0x370...158ea
transact to C.updateData pending ...
[vm] from: 0x583...eddC4 to: C.updateData(uint256) 0xd91...39138 value: 0 wei data: 0xd9c...00007 logs: 0
hash: 0xdc2...cd09b
call to C.getData
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: C.getData() data: 0x3bc...50e30
Debug

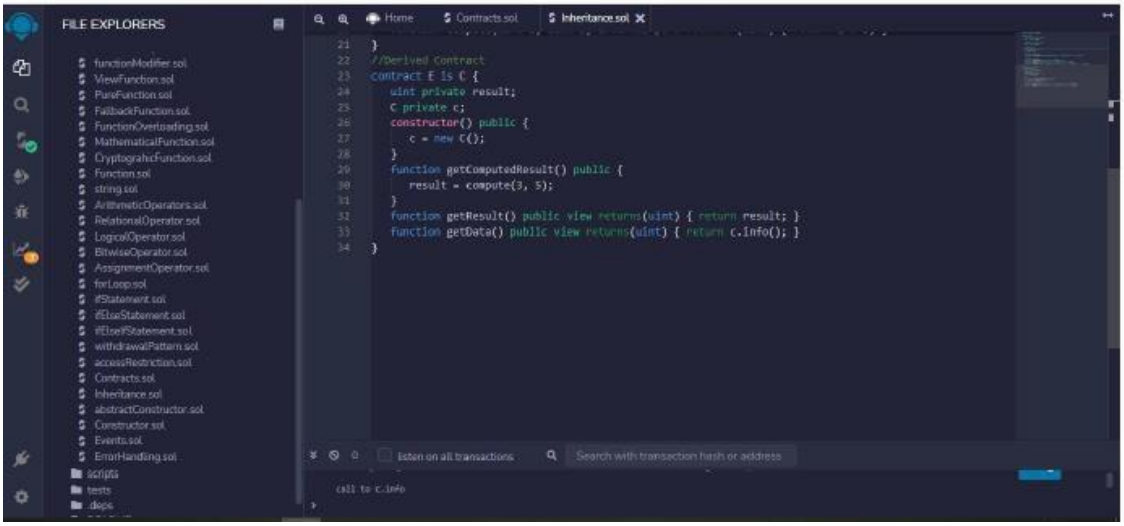
Inheritance:

Code:

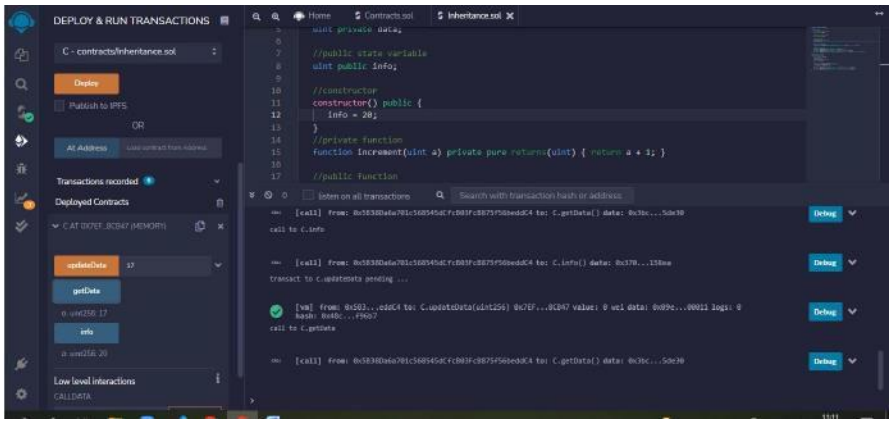
```
pragma solidity ^0.8.0;
/// simple inheritance example with animals
/// @notice Base contract representing a generic animal
contract Animal {
    string public name;
    constructor(string memory _name) {
        name = _name;
    }

    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!";
    }
}
```





Output:



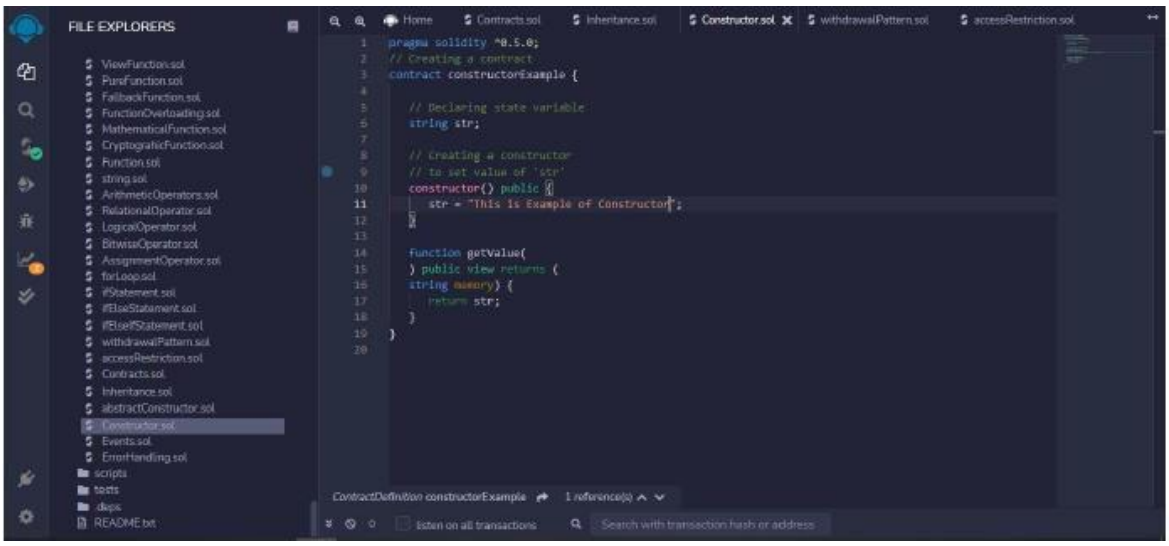
Constructors:

Code:

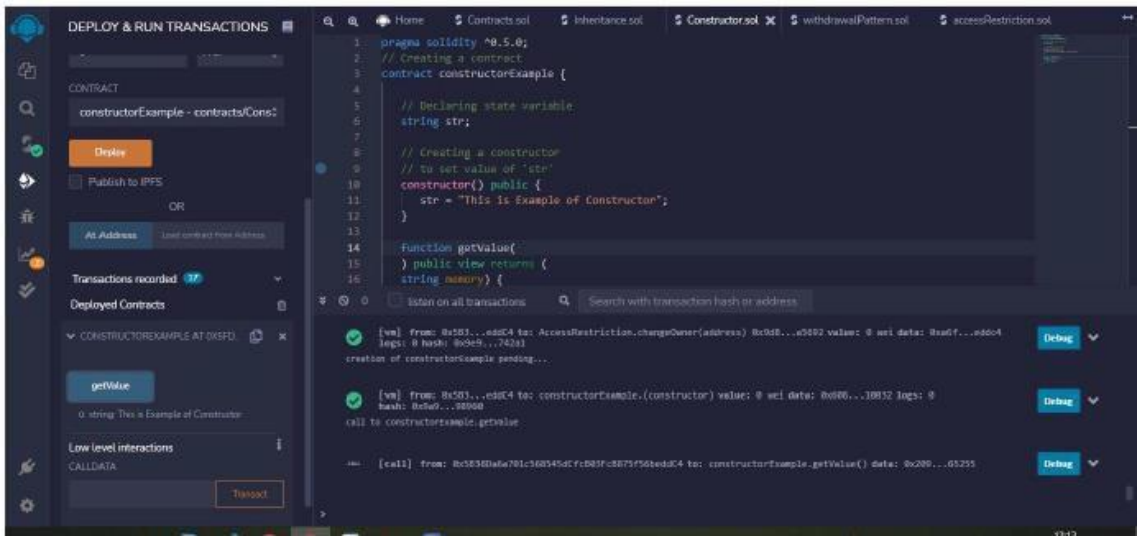
```
pragma solidity ^0.8.0;

/// @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);
    }
}
```



Output:



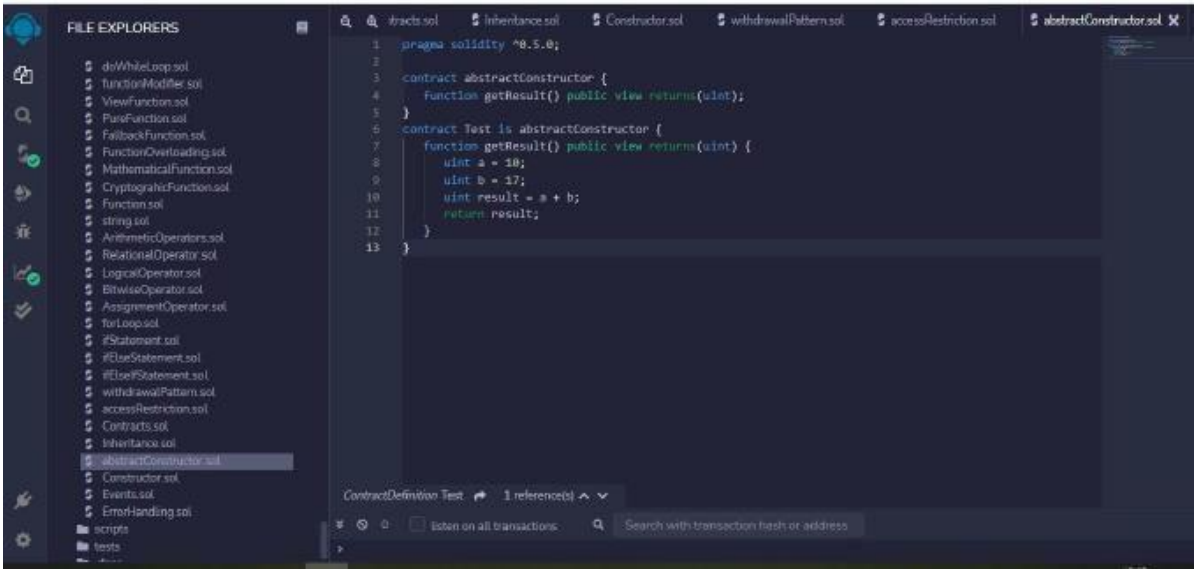
Abstract Contracts:

Code:

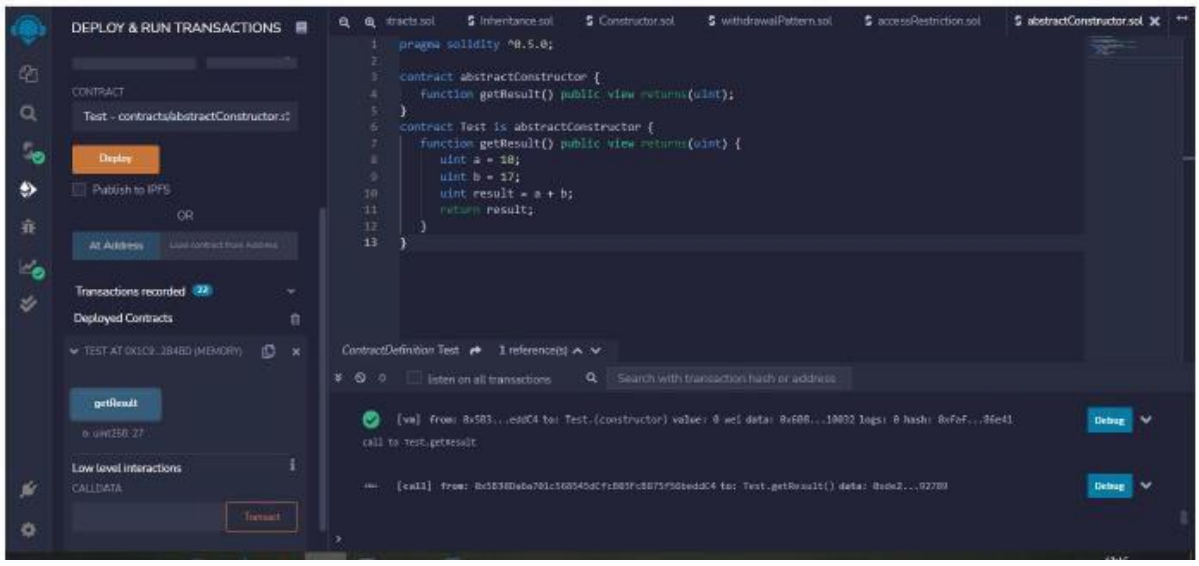
```
pragma solidity ^0.8.0;

/// @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;
    }
}
```



Output:

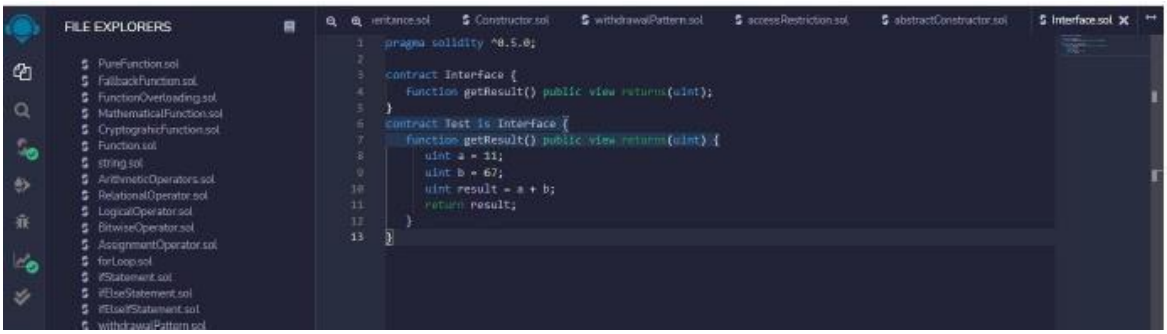


Interface:

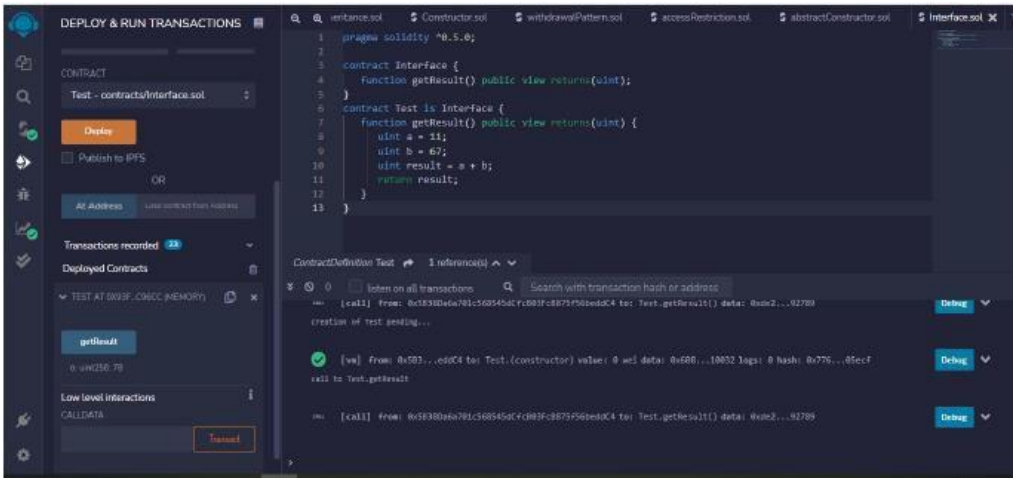
Code:

```
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);
    }
}
```



Output:



PRACTICAL 3 D.

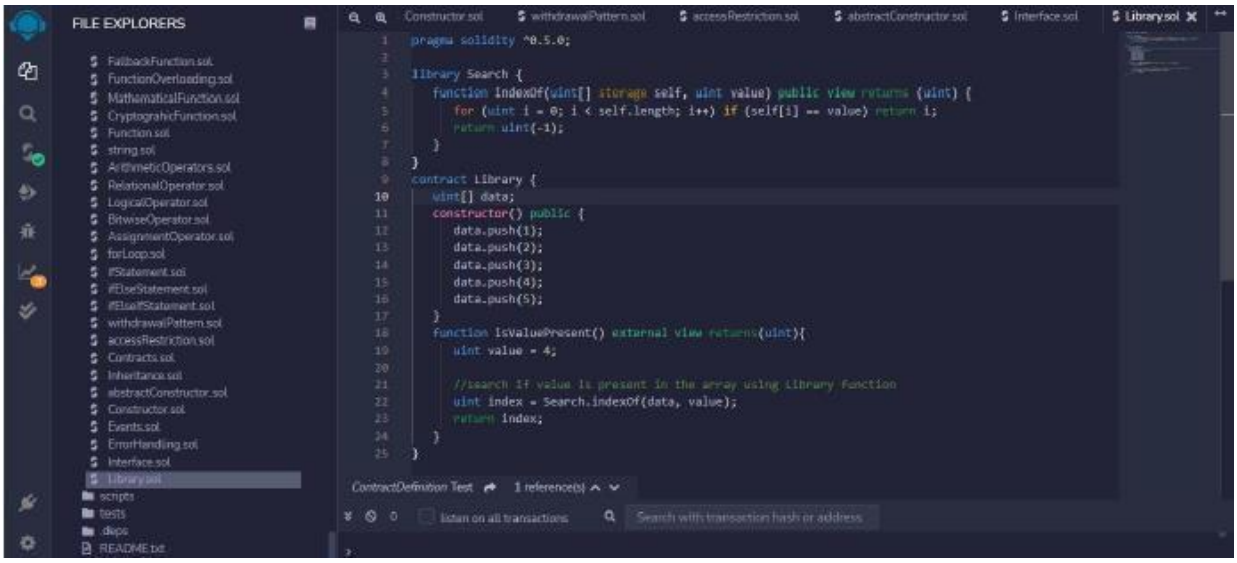
Aim: Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling

Libraries:

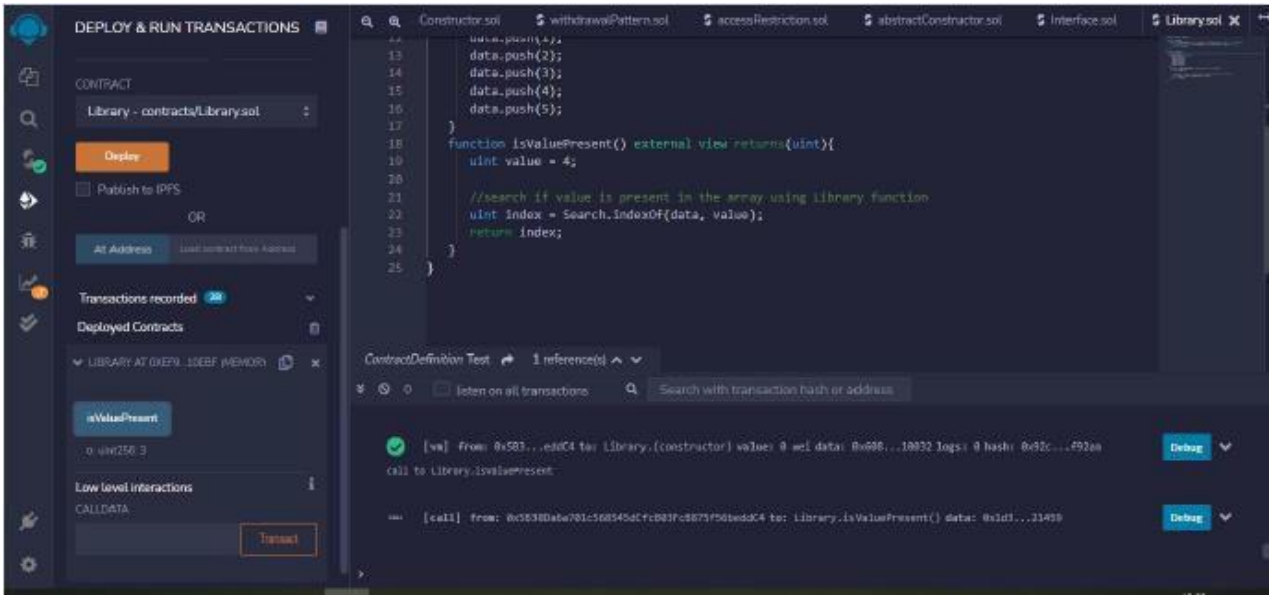
Code:

```
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);
    }
}
```



Output:

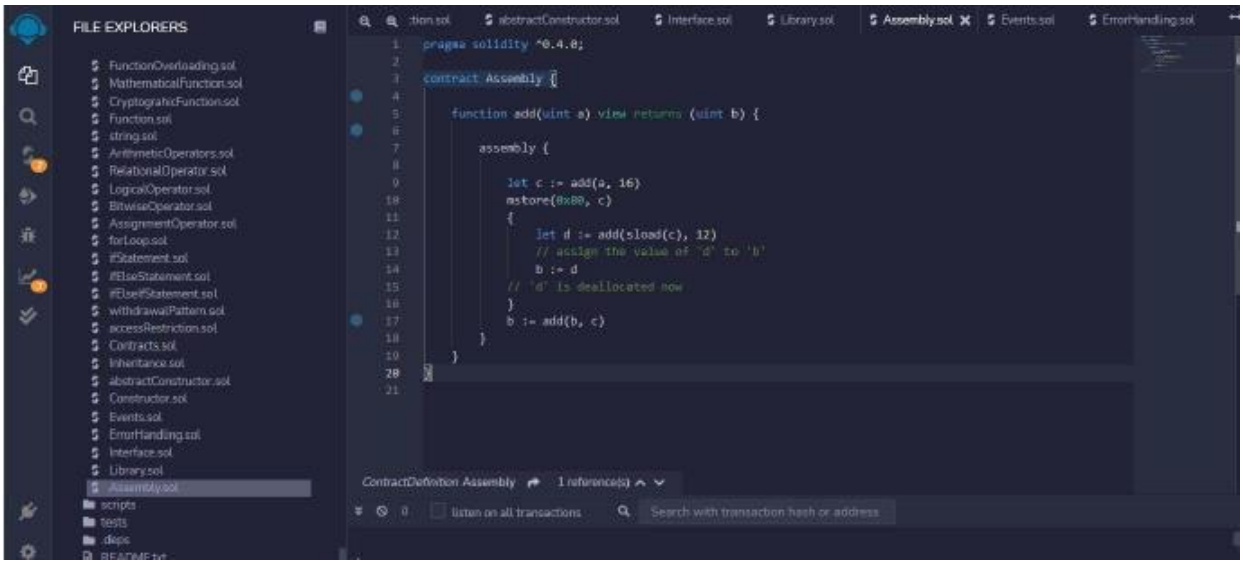


Assembly:

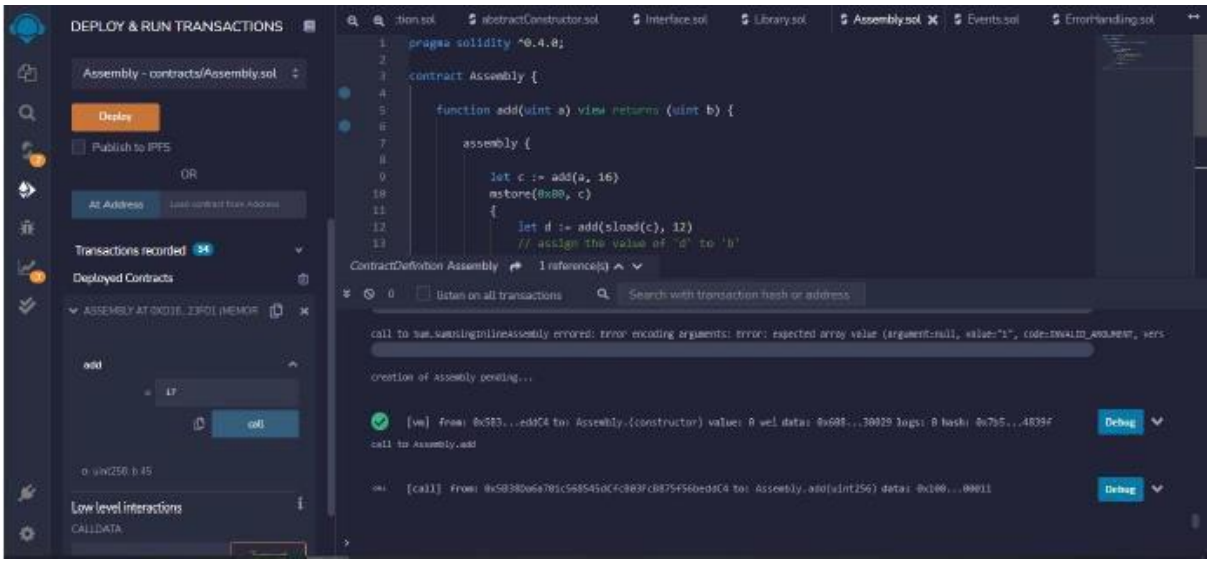
Code:

```
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
            }
        }
    }
}
```



Output:

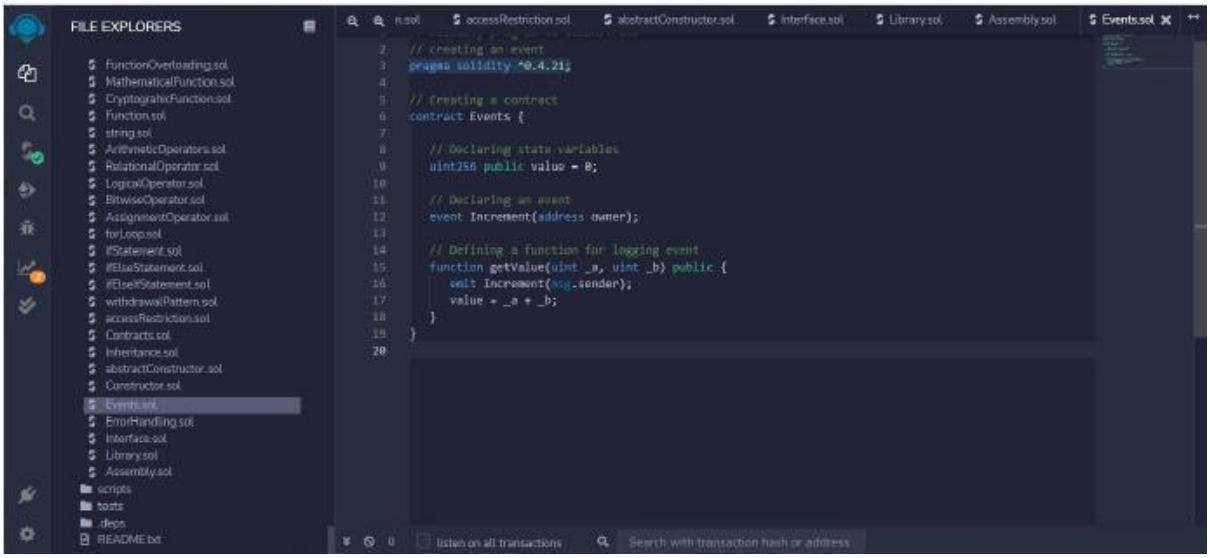


Events:

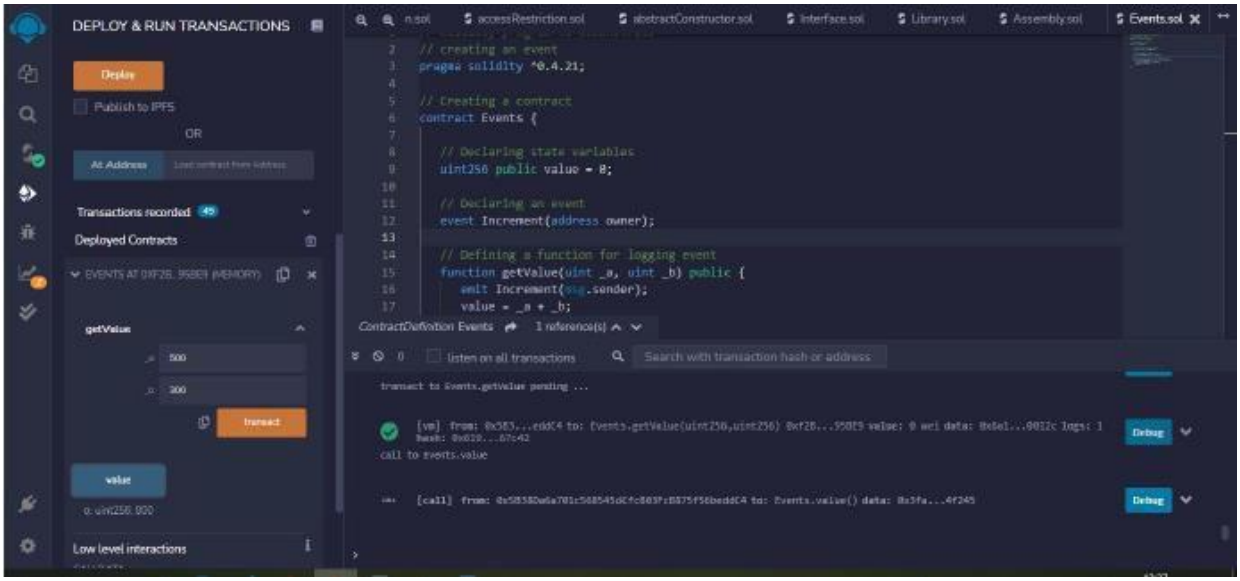
Code:

```
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]);
    }
}
```


FILE EXPLORERS
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
IfElseIfStatement.sol
withdrawalPattern.sol
accessRestriction.sol
Contracts.sol
Inheritance.sol
abstractConstructor.sol
Constructor.sol
Events.sol
ErrorHandling.sol
Interface.sol
Library.sol
Assembly.sol
scripts
tests
.deps
README.txt
// creating an event
pragma solidity ^0.4.21;
// Creating a contract
contract Events {
// Declaring state variables
uint256 public value = 0;
// Declaring an event
event Increment(address owner);
// Defining a function for logging event
function getValue(uint _a, uint _b) public {
emit Increment(msg.sender);
value = _a + _b;
}
}
listen on all transactions
Search with transaction hash or address

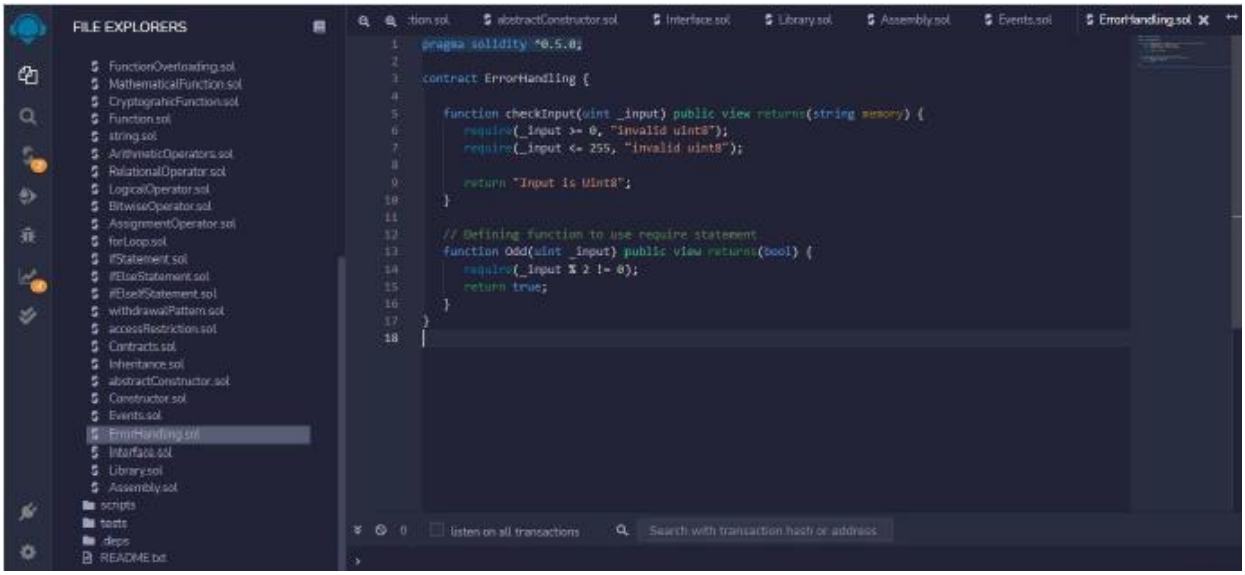
Output:


DEPLOY & RUN TRANSACTIONS
Deploy
Publish to IPFS
OR
At Address
Transactions recorded 45
Deployed Contracts
getValue
500
300
transact
value
0: uint256: 800
Low level interactions
// creating an event
pragma solidity ^0.4.21;
// Creating a contract
contract Events {
// Declaring state variables
uint256 public value = 0;
// Declaring an event
event Increment(address owner);
// Defining a function for logging event
function getValue(uint _a, uint _b) public {
emit Increment(msg.sender);
value = _a + _b;
ContractDefinition Events 1 reference(s)
listen on all transactions
Search with transaction hash or address
transact to Events.getValue pending ...
call to Events.value
Debug
Debug

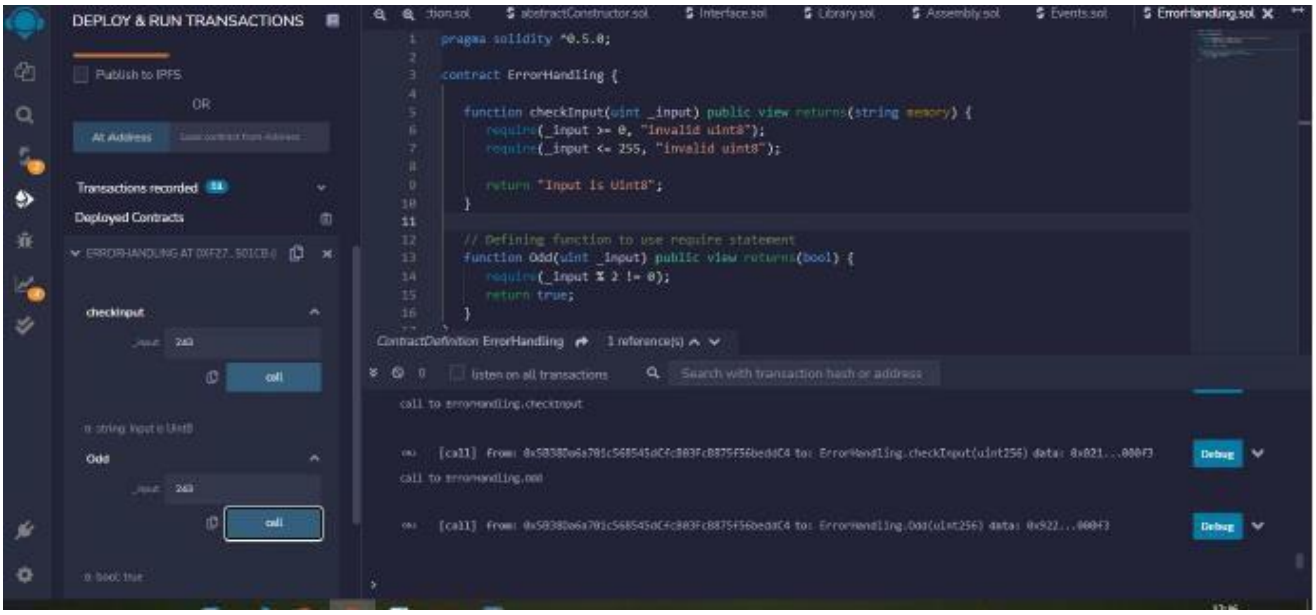
Error Handling:

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);
    }
}
```


FILE EXPLORERS
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
IfElseIfStatement.sol
withdrawalPattern.sol
accessRestriction.sol
Contracts.sol
Inheritance.sol
abstractConstructor.sol
Constructor.sol
Events.sol
ErrorHandling.sol
Interface.sol
Library.sol
Assembly.sol
scripts
tests
deps
README.txt
pragma solidity ^0.5.0;
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;
}
}
listen on all transactions
Search with transaction hash or address

Output:


DEPLOY & RUN TRANSACTIONS
Publish to IPFS
OR
At Address
Transactions recorded
Deployed Contracts
checkInput
call
Odd
call
pragma solidity ^0.5.0;
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
listen on all transactions
Search with transaction hash or address
call to errorHandling.checkInput
call to errorHandling.Odd
Debug
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-Irhoa

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.
   - Install Docker from Docker's official site.
2. **Docker Compose**: This tool allows you to define and run multi-container Docker applications.
   - Install Docker Compose from here.
3. **Git**: To clone repositories.
   - Install Git from here.
4. **Java**: Hyperledger Iroha is developed using Java, so make sure you have Java (JDK 11 or higher) installed.
   - Install Java from here.
5. **CMake and build tools** (for building the Hyperledger Iroha binaries, if you plan to build it yourself):
   - Install CMake from here.
   - Install build-essential (for Linux) via the package manager:
   - 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**:
   - For Ubuntu:
   - sudo apt-get update
   - sudo apt-get install -y cmake g++ libssl-dev libboost-all-dev \
   - libgmp-dev libsodium-dev python3-pip libcurl4-openssl-dev \
   - 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": {
```

```
      "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.

NURTURING POTENTIAL

SAKET GYANPEETH'S

SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Permanently Affiliated to University of Mumbai)

NAAC Accredited

Saket Vidyanagri Marg, Chinchpada Road, Katemanivali,
Kalyan (East) -421306(Mah)

Department of Information Technology

This is to certify that

Mr. SACHIDANAND SANTOSH DUBEY

Seat No. 1314296

of

M.Sc. Information Technology

Part II NEP 2020 Semester IV

has satisfactory carried out the required practical in the subject of **DEEP LEARNING**
For the Academic year 2024 – 2025

Practical In-Charge **Head of the Department**

External Examiner

College Seal

INDEX

Sr. No.		Practical	Signature
1.	a.	• Create tensors with different shapes and data types. • Perform basic operations like addition, subtraction, multiplication, and division on tensors. • Reshape, slice, and index tensors to extract specific elements or sections. • Performing matrix multiplication and finding eigenvectors and eigenvalues using Tensor Flow	
	b.	Program to solve the XOR problem.	
2.	a.	• Implement a simple linear regression model using TensorFlow's lowlevel API (or tf. keras). • Train the model on a toy dataset (e.g., housing prices vs. square footage). • Visualize the loss function and the learned linear relationship. • Make predictions on new data points.	
3.	a.	Implementing deep neural network for performing binary classification task	
	b.	Using a deep feed-forward network with two hidden layers for performing multiclass classification and predicting the class.	
4.		Write a program to implement deep learning Techniques for image segmentation	
5.		Write a program to predict a caption for a sample image using LSTM.	

6.	Applying the Autoencoder algorithms for encoding real-world data	
7.	Write a program for character recognition using RNN and compare it with CNN.	
8.	Write a program to develop Autoencoders using MNIST Handwritten Digits.	
9.	Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.(google stock price).	
10.	Applying Generative Adversarial Networks for image generation and unsupervised tasks.	

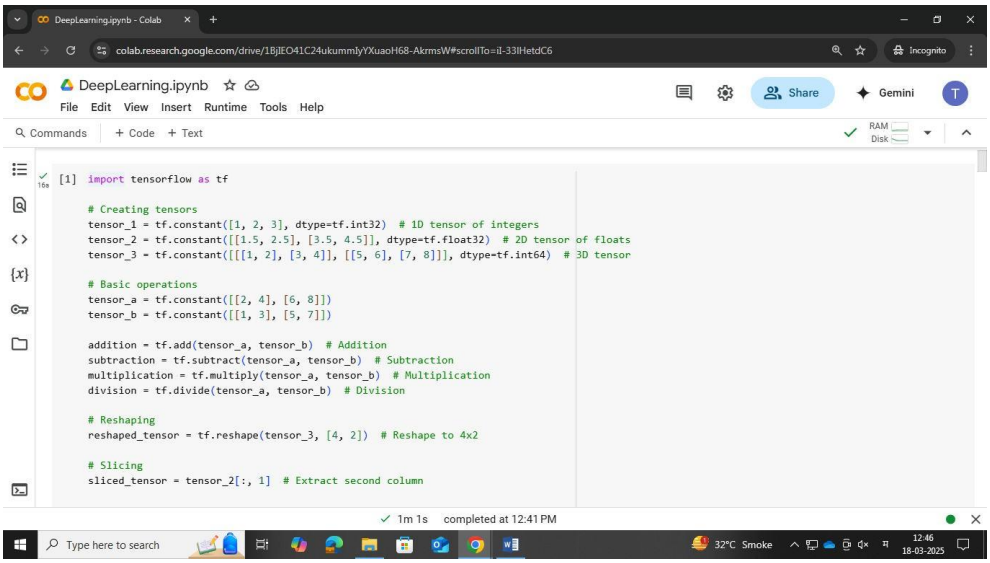
PRACTICAL 1 A.

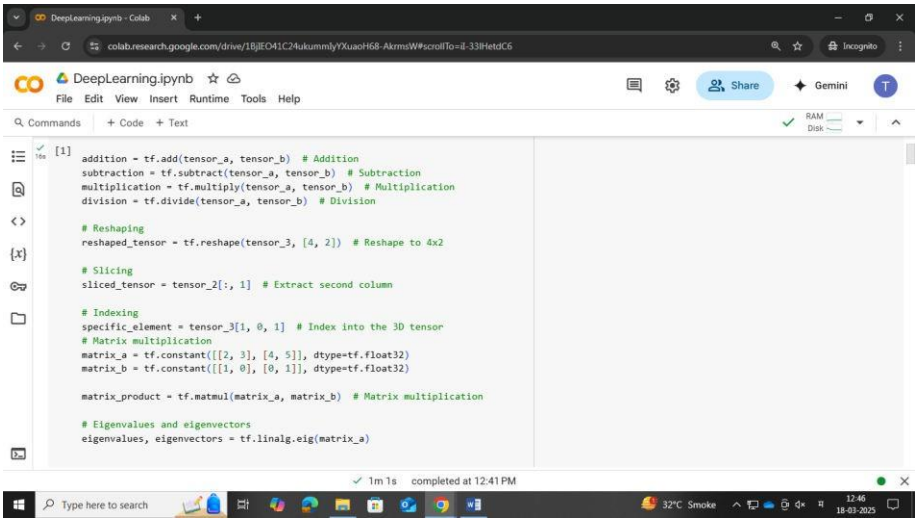
Aim:

- Create tensors with different shapes and data types.
- Perform basic operations like addition, subtraction, multiplication, and division on tensors.
- Reshape, slice, and index tensors to extract specific elements or sections.
- Performing matrix multiplication and finding eigenvectors and eigenvalues using Tensor Flow

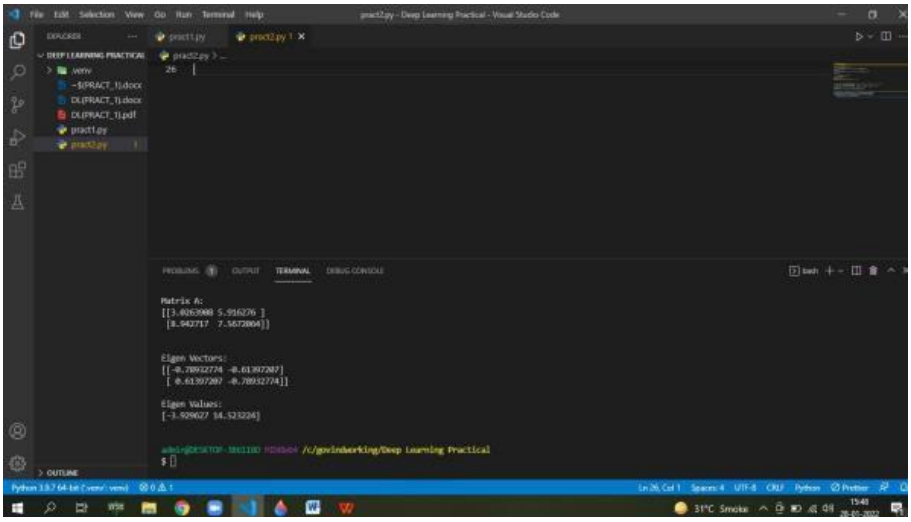
Code:

```
import tensorflow as tf
# Creating tensors
tensor_1 = tf.constant([1, 2, 3], dtype=tf.int32)    # 1D tensor of integers
tensor_2 = tf.constant([[1.5, 2.5], [3.5, 4.5]], dtype=tf.float32)    # 2D
tensor of floats
tensor_3 = tf.constant([[[1, 2], [3, 4]], [[5, 6], [7, 8]]],
dtype=tf.int64)   # 3D tensor

# Basic operations
tensor_a = tf.constant([[2, 4], [6, 8]])
tensor_b = tf.constant([[1, 3], [5, 7]])

addition = tf.add(tensor_a, tensor_b)    # Addition
subtraction = tf.subtract(tensor_a, tensor_b)    # Subtraction
multiplication = tf.multiply(tensor_a, tensor_b)    # Multiplication
division = tf.divide(tensor_a, tensor_b)    # Division
# Reshaping
reshaped_tensor = tf.reshape(tensor_3, [4, 2])    # Reshape to 4x2
# Slicing
sliced_tensor = tensor_2[:, 1]   # Extract second column

# Indexing
specific_element = tensor_3[1, 0, 1]    # Index into the 3D tensor
# Matrix multiplication
matrix_a = tf.constant([[2, 3], [4, 5]], dtype=tf.float32)
matrix_b = tf.constant([[1, 0], [0, 1]], dtype=tf.float32)

matrix_product = tf.matmul(matrix_a, matrix_b)   # Matrix multiplication

# Eigenvalues and eigenvectors
eigenvalues, eigenvectors = tf.linalg.eig(matrix_a)
```


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummlyYXuaoH68-AkrmsW#scrollTo=iI-33IHetdC6
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
[1] import tensorflow as tf
# Creating tensors
tensor_1 = tf.constant([1, 2, 3], dtype=tf.int32) # 1D tensor of integers
tensor_2 = tf.constant([[1.5, 2.5], [3.5, 4.5]], dtype=tf.float32) # 2D tensor of floats
tensor_3 = tf.constant([[[1, 2], [3, 4]], [[5, 6], [7, 8]]], dtype=tf.int64) # 3D tensor
# Basic operations
tensor_a = tf.constant([[2, 4], [6, 8]])
tensor_b = tf.constant([[1, 3], [5, 7]])
addition = tf.add(tensor_a, tensor_b) # Addition
subtraction = tf.subtract(tensor_a, tensor_b) # Subtraction
multiplication = tf.multiply(tensor_a, tensor_b) # Multiplication
division = tf.divide(tensor_a, tensor_b) # Division
# Reshaping
reshaped_tensor = tf.reshape(tensor_3, [4, 2]) # Reshape to 4x2
# Slicing
sliced_tensor = tensor_2[:, 1] # Extract second column
1m 1s completed at 12:41 PM
Type here to search
32°C Smoke
12:46
18-03-2025


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummlyYXuaoH68-AkrmsW#scrollTo=iI-33IHetdC6
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
[1] addition = tf.add(tensor_a, tensor_b) # Addition
subtraction = tf.subtract(tensor_a, tensor_b) # Subtraction
multiplication = tf.multiply(tensor_a, tensor_b) # Multiplication
division = tf.divide(tensor_a, tensor_b) # Division
# Reshaping
reshaped_tensor = tf.reshape(tensor_3, [4, 2]) # Reshape to 4x2
# Slicing
sliced_tensor = tensor_2[:, 1] # Extract second column
# Indexing
specific_element = tensor_3[1, 0, 1] # Index into the 3D tensor
# Matrix multiplication
matrix_a = tf.constant([[2, 3], [4, 5]], dtype=tf.float32)
matrix_b = tf.constant([[1, 0], [0, 1]], dtype=tf.float32)
matrix_product = tf.matmul(matrix_a, matrix_b) # Matrix multiplication
# Eigenvalues and eigenvectors
eigenvalues, eigenvectors = tf.linalg.eig(matrix_a)
1m 1s completed at 12:41 PM
Type here to search
32°C Smoke
12:46
18-03-2025

Output:

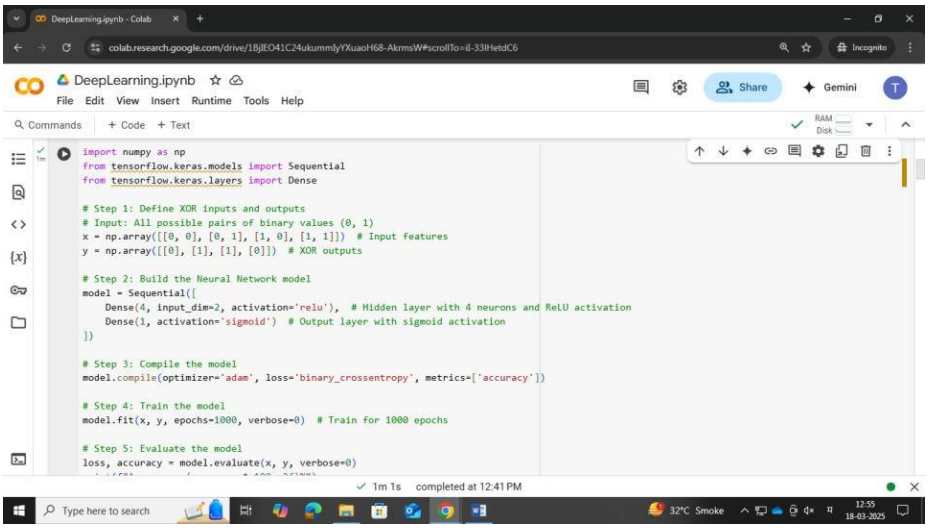

pract1.py
pract2.py
DEEP LEARNING PRACTICAL
venv
DL(PRACT_1).docx
DL(PRACT_1).pdf
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
Matrix A:
[[3.0263988 5.916276]
[8.940717 7.5672004]]
Eigen Vectors:
[[-0.78932774 -0.61397207]
[0.61397207 -0.78932774]]
Eigen Values:
[-1.929027 14.523224]
$

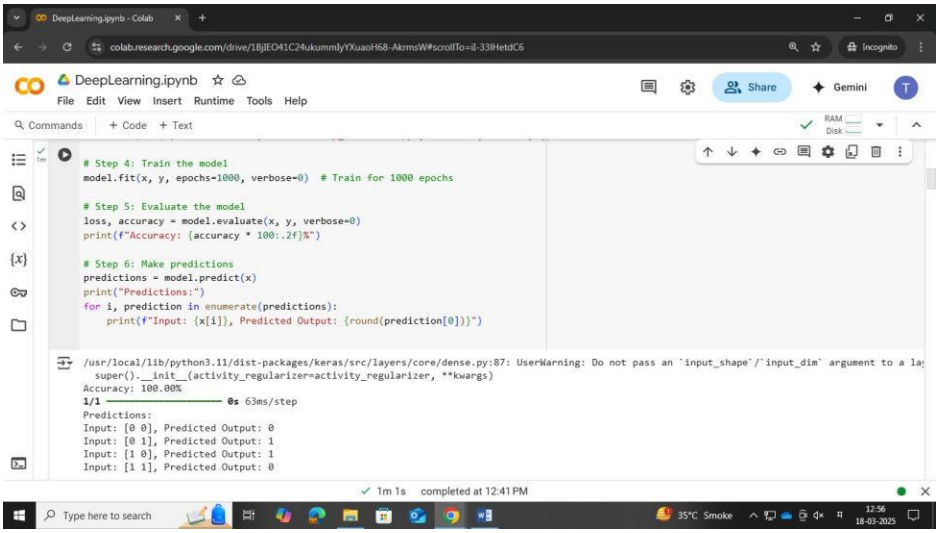
PRACTICAL 1 B.

Aim: Program to solve the XOR problem.

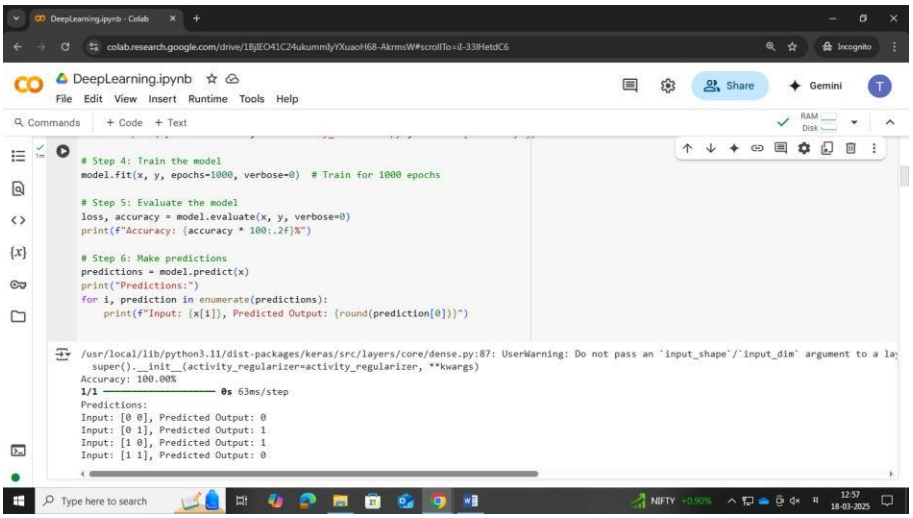
Code:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Step 1: Define XOR inputs and outputs
# Input: All possible pairs of binary values (0, 1)
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])   # Input features
y = np.array([[0], [1], [1], [0]])   # XOR outputs

# Step 2: Build the Neural Network model
model = Sequential([
    Dense(4, input_dim=2, activation='relu'),   # Hidden layer with 4
neurons and ReLU activation
    Dense(1, activation='sigmoid')   # Output layer with sigmoid activation
])

#  Step 3: Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Step 4: Train the model
model.fit(x, y, epochs=1000, verbose=0)   # Train for 1000 epochs

# Step 5: Evaluate the model
loss, accuracy = model.evaluate(x, y, verbose=0)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Step 6: Make predictions
predictions = model.predict(x)
print("Predictions:")
for i, prediction in enumerate(predictions):
    print(f"Input: {x[i]}, Predicted Output: {round(prediction[0])}")
```


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummlyYXuaoH68-AkrmsW#scrollTo=il-33IHetdC6
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Step 1: Define XOR inputs and outputs
# Input: All possible pairs of binary values (0, 1)
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Input features
y = np.array([[0], [1], [1], [0]]) # XOR outputs
# Step 2: Build the Neural Network model
model = Sequential([
Dense(4, input_dim=2, activation='relu'), # Hidden layer with 4 neurons and ReLU activation
Dense(1, activation='sigmoid') # Output layer with sigmoid activation
])
# Step 3: Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Step 4: Train the model
model.fit(x, y, epochs=1000, verbose=0) # Train for 1000 epochs
Step 5: Evaluate the model
loss, accuracy = model.evaluate(x, y, verbose=0)
1m 1s completed at 12:41 PM
Type here to search
32°C Smoke
12:55
18-03-2025


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummlyYXuaoH68-AkrmsW#scrollTo=il-33IHetdC6
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Step 4: Train the model
model.fit(x, y, epochs=1000, verbose=0) # Train for 1000 epochs
Step 5: Evaluate the model
loss, accuracy = model.evaluate(x, y, verbose=0)
print(f"Accuracy: {accuracy * 100:.2f}%")
# Step 6: Make predictions
predictions = model.predict(x)
print("Predictions:")
for i, prediction in enumerate(predictions):
print(f"Input: {x[i]}, Predicted Output: {round(prediction[0])}")
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Accuracy: 100.00%
1/1 0s 63ms/step
Predictions:
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 0
1m 1s completed at 12:41 PM
Type here to search
35°C Smoke
12:56
18-03-2025

Output:


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummlyYXuaoH68-AkrmsW#scrollTo=il-33IHetdC6
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Step 4: Train the model
model.fit(x, y, epochs=1000, verbose=0) # Train for 1000 epochs
Step 5: Evaluate the model
loss, accuracy = model.evaluate(x, y, verbose=0)
print(f"Accuracy: {accuracy * 100:.2f}%")
# Step 6: Make predictions
predictions = model.predict(x)
print("Predictions:")
for i, prediction in enumerate(predictions):
print(f"Input: {x[i]}, Predicted Output: {round(prediction[0])}")
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Accuracy: 100.00%
1/1 0s 63ms/step
Predictions:
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 0
Type here to search
NIFTY +0.90%
12:57
18-03-2025

PRACTICAL 2 A.

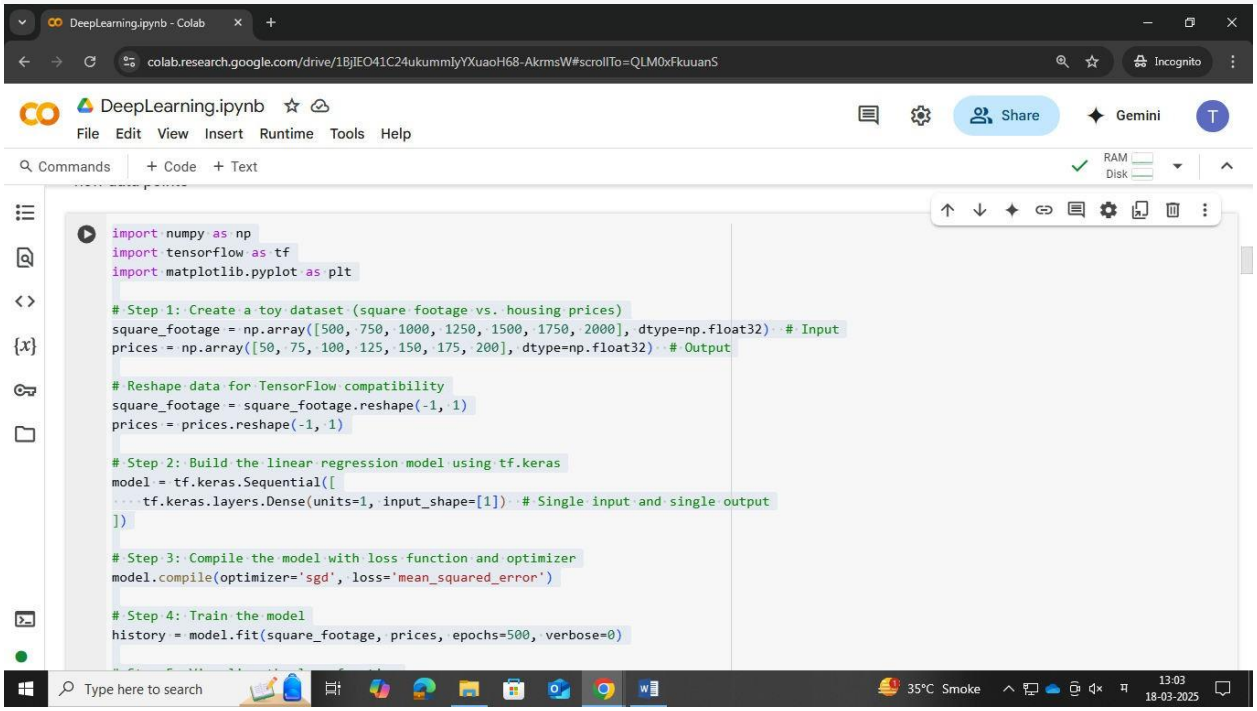
Aim:

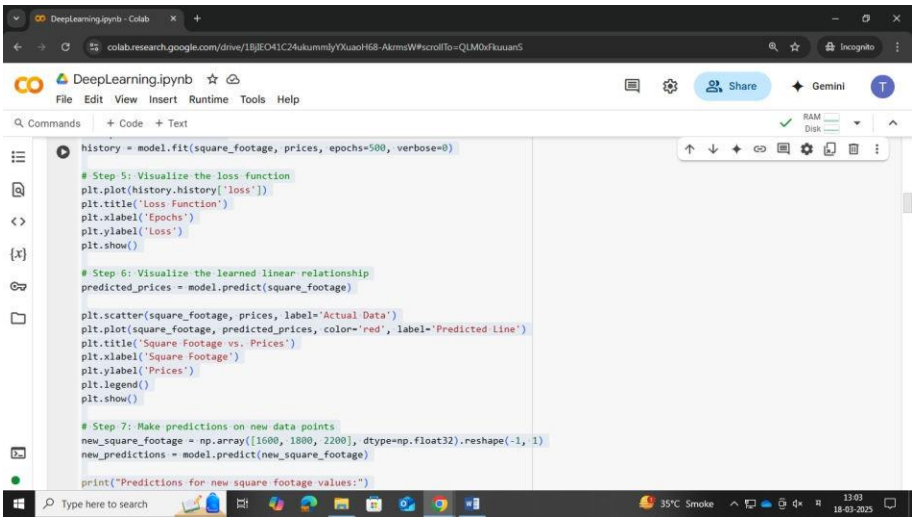
- Implement a simple linear regression model using TensorFlow's lowlevel API (or tf. keras).
- Train the model on a toy dataset (e.g., housing prices vs. square footage).
- Visualize the loss function and the learned linear relationship.
- Make predictions on new data points.

Code:

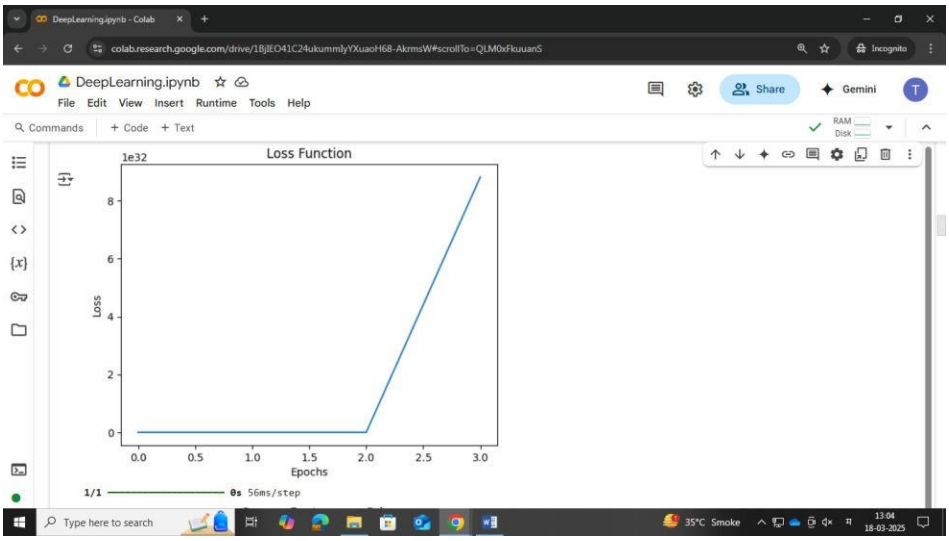
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Step 1: Create a toy dataset (square footage vs. housing prices)
square_footage = np.array([500, 750, 1000, 1250, 1500, 1750, 2000],
dtype=np.float32)  # Input
prices = np.array([50, 75, 100, 125, 150, 175, 200], dtype=np.float32)  #
Output

# Reshape data for TensorFlow compatibility
square_footage = square_footage.reshape(-1, 1)
prices = prices.reshape(-1, 1)

# Step 2: Build the linear regression model using tf.keras
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=[1])  # Single input and
single output
])

# Step 3: Compile the model with loss function and optimizer
model.compile(optimizer='sgd', loss='mean_squared_error')

# Step 4: Train the model
history = model.fit(square_footage, prices, epochs=500, verbose=0)

# Step 5: Visualize the loss function
plt.plot(history.history['loss'])
plt.title('Loss Function')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

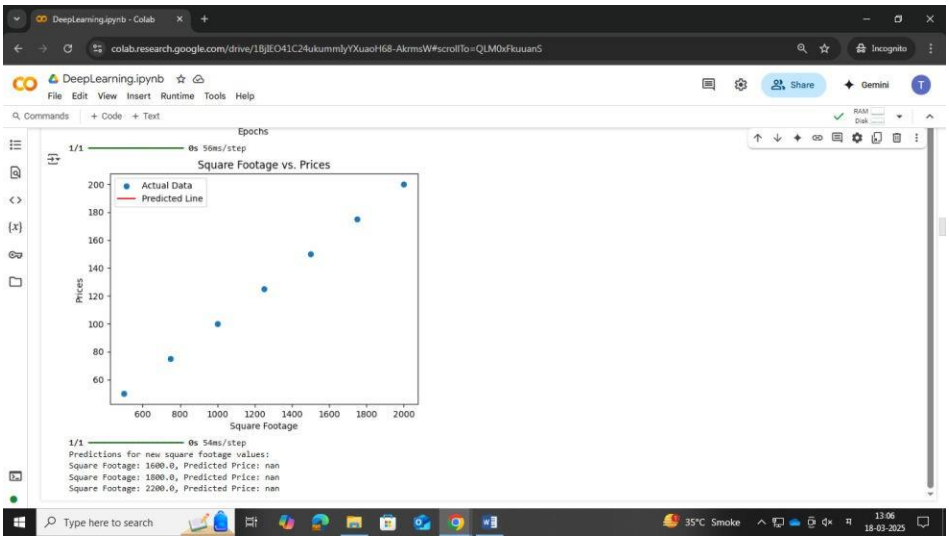
```
# Step 6: Visualize the learned linear relationship
predicted_prices = model.predict(square_footage)

plt.scatter(square_footage, prices, label='Actual Data')
plt.plot(square_footage, predicted_prices, color='red', label='Predicted
Line')
plt.title('Square Footage vs. Prices')
plt.xlabel('Square Footage')
plt.ylabel('Prices')
plt.legend()
plt.show()

# Step 7: Make predictions on new data points
new_square_footage = np.array([1600, 1800, 2200],
dtype=np.float32).reshape(-1, 1)
new_predictions = model.predict(new_square_footage)

print("Predictions for new square footage values:")
for i, sqft in enumerate(new_square_footage):
    print(f"Square Footage: {sqft[0]}, Predicted Price:
{new_predictions[i][0]:.2f}")
```




DeepLearning.ipynb - Colab
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
history = model.fit(square_footage, prices, epochs=500, verbose=0)
# Step 5: Visualize the loss function
plt.plot(history.history['loss'])
plt.title('Loss Function')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
# Step 6: Visualize the learned linear relationship
predicted_prices = model.predict(square_footage)
plt.scatter(square_footage, prices, label='Actual Data')
plt.plot(square_footage, predicted_prices, color='red', label='Predicted Line')
plt.title('Square Footage vs. Prices')
plt.xlabel('Square Footage')
plt.ylabel('Prices')
plt.legend()
plt.show()
# Step 7: Make predictions on new data points
new_square_footage = np.array([1600, 1800, 2200], dtype=np.float32).reshape(-1, 1)
new_predictions = model.predict(new_square_footage)
print("Predictions for new square footage values:")

Output ;


DeepLearning.ipynb - Colab
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Loss Function
1e32
Loss
Epochs
1/1 0s 56ms/step


DeepLearning.ipynb - Colab
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Epochs
1/1 0s 56ms/step
Square Footage vs. Prices
Actual Data
Predicted Line
Prices
Square Footage
1/1 0s 54ms/step
Predictions for new square footage values:
Square Footage: 1600.0, Predicted Price: nan
Square Footage: 1800.0, Predicted Price: nan
Square Footage: 2200.0, Predicted Price: nan

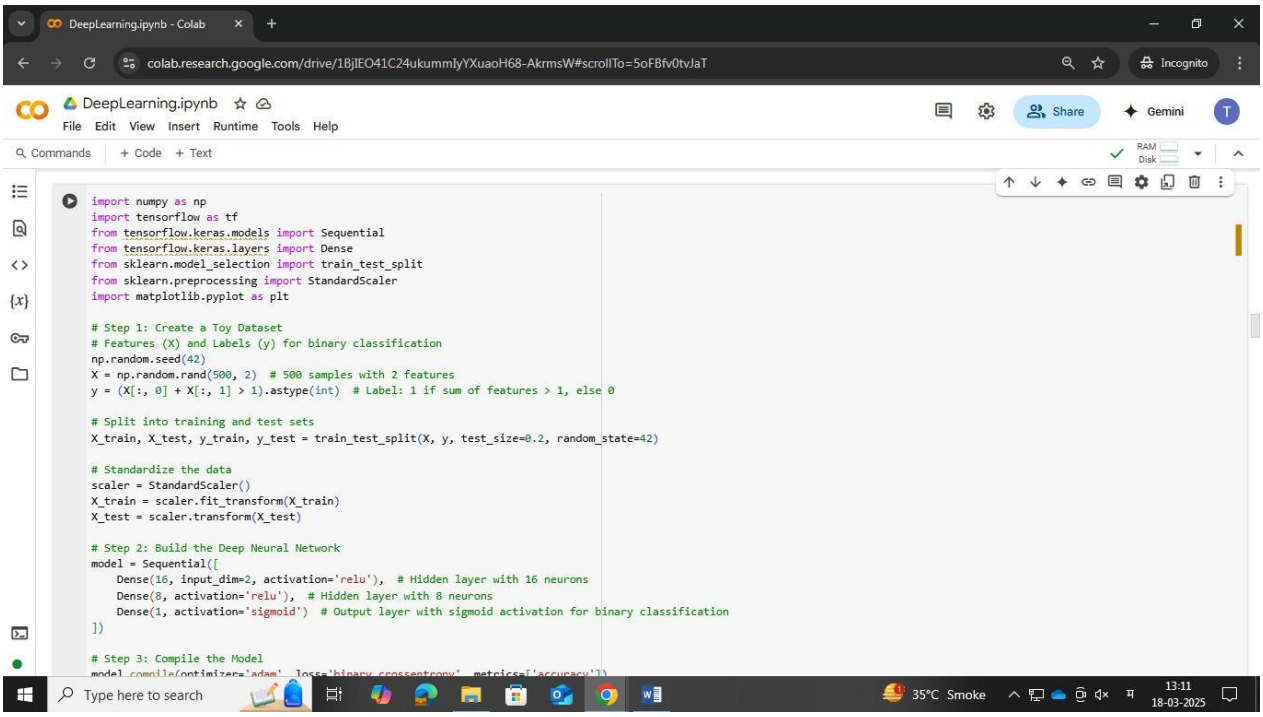
PRACTICAL 3 A.

Aim: Implementing deep neural network for performing binary classification task

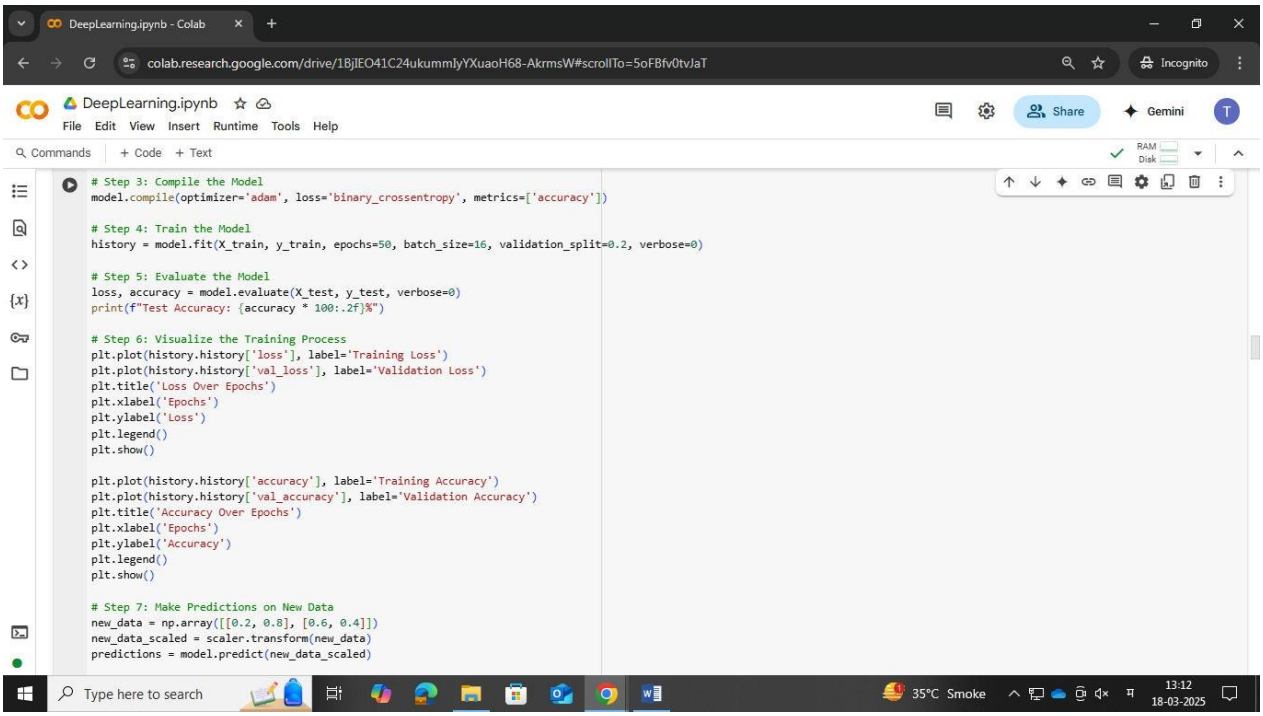
Code:

```
import numpy as np

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Step 1: Create a Toy Dataset
# Features (X) and Labels (y) for binary classification
np.random.seed(42)
X = np.random.rand(500, 2)  # 500 samples with 2 features
y = (X[:, 0] + X[:, 1] > 1).astype(int)     # Label: 1 if sum of features >
1, else 0

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 2: Build the Deep Neural Network
model = Sequential([
    Dense(16, input_dim=2, activation='relu'),   # Hidden layer with 16
neurons
    Dense(8, activation='relu'),   # Hidden layer with 8 neurons
    Dense(1, activation='sigmoid')   # Output layer with sigmoid activation
for binary classification
])

#  Step 3: Compile the Model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Step 4: Train the Model
```

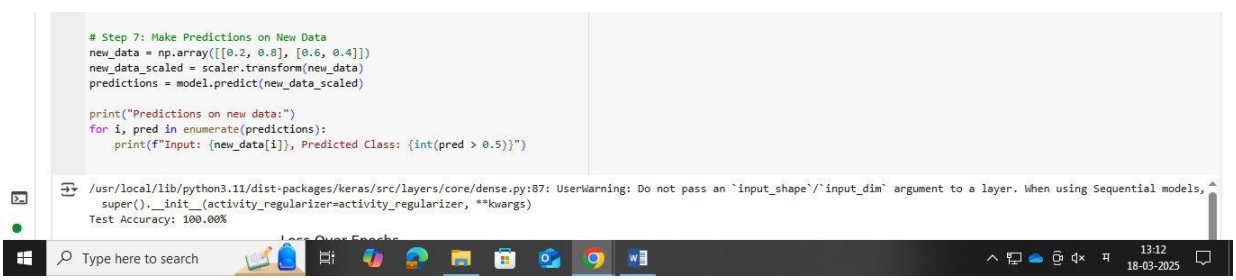
```
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
validation_split=0.2, verbose=0)

# Step 5: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Step 6: Visualize the Training Process
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Step 7: Make Predictions on New Data
new_data = np.array([[0.2, 0.8], [0.6, 0.4]])
new_data_scaled = scaler.transform(new_data)
predictions = model.predict(new_data_scaled)

print("Predictions on new data:")
for i, pred in enumerate(predictions):
    print(f"Input: {new_data[i]}, Predicted Class: {int(pred > 0.5)}")
```



```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Step 1: Create a Toy Dataset
# Features (X) and Labels (y) for binary classification
np.random.seed(42)
X = np.random.rand(500, 2)  # 500 samples with 2 features
y = (X[:, 0] + X[:, 1] > 1).astype(int)  # Label: 1 if sum of features > 1, else 0

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 2: Build the Deep Neural Network
model = Sequential([
    Dense(16, input_dim=2, activation='relu'),  # Hidden layer with 16 neurons
    Dense(8, activation='relu'),  # Hidden layer with 8 neurons
    Dense(1, activation='sigmoid')  # Output layer with sigmoid activation for binary classification
])

# Step 3: Compile the Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```



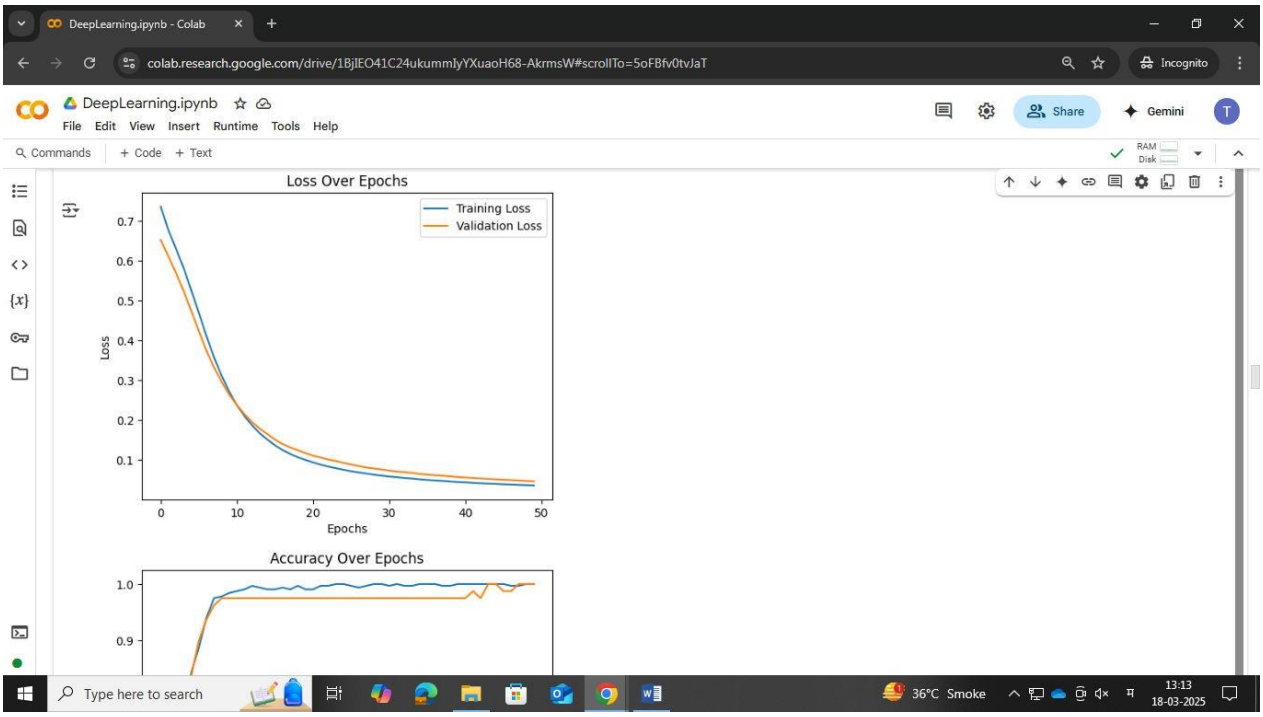
```
# Step 3: Compile the Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 4: Train the Model
history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_split=0.2, verbose=0)

# Step 5: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Step 6: Visualize the Training Process
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Step 7: Make Predictions on New Data
new_data = np.array([[0.2, 0.8], [0.6, 0.4]])
new_data_scaled = scaler.transform(new_data)
predictions = model.predict(new_data_scaled)
```



```
# Step 7: Make Predictions on New Data
new_data = np.array([[0.2, 0.8], [0.6, 0.4]])
new_data_scaled = scaler.transform(new_data)
predictions = model.predict(new_data_scaled)

print("Predictions on new data:")
for i, pred in enumerate(predictions):
    print(f"Input: {new_data[i]}, Predicted Class: {int(pred > 0.5)}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Test Accuracy: 100.00%
```

Output:


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummIyYXuaoH68-AkrmsW#scrollTo=5oFBfv0tvJaT
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Loss Over Epochs
Training Loss
Validation Loss
Loss
Epochs
Accuracy Over Epochs
Type here to search
36°C Smoke
13:13
18-03-2025


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummIyYXuaoH68-AkrmsW#scrollTo=5oFBfv0tvJaT
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Accuracy Over Epochs
Training Accuracy
Validation Accuracy
Accuracy
Epochs
1/1 0s 79ms/step
Predictions on new data:
Input: [0.2 0.8], Predicted Class: 0
Input: [0.6 0.4], Predicted Class: 0
<ipython-input-14-f64110c0dcf4>:64: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single eleme
print(f"Input: {new_data[i]}, Predicted Class: {int(pred > 0.5)}")
Type here to search
36°C Smoke
13:13
18-03-2025

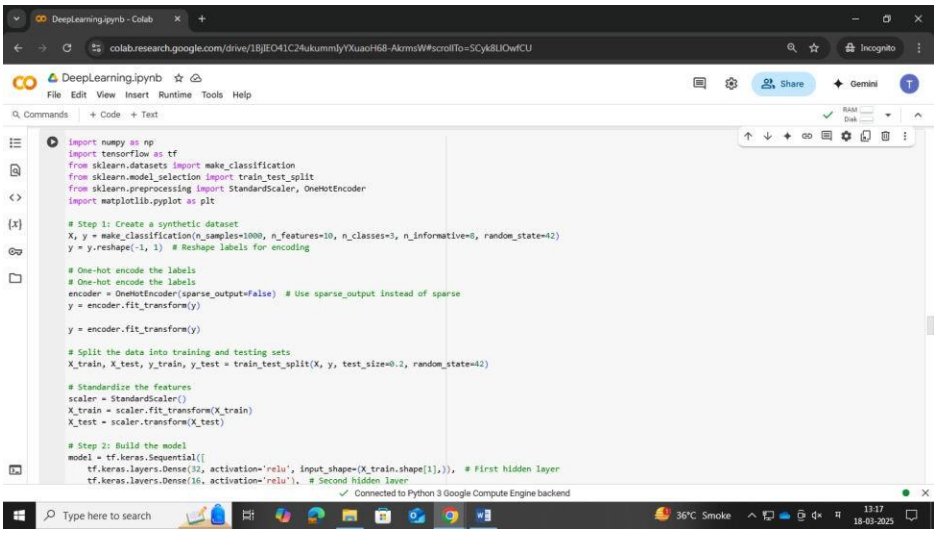
PRACTICAL 3 B.

Aim: Using a deep feed-forward network with two hidden layers for performing multiclass classification and predicting the class.

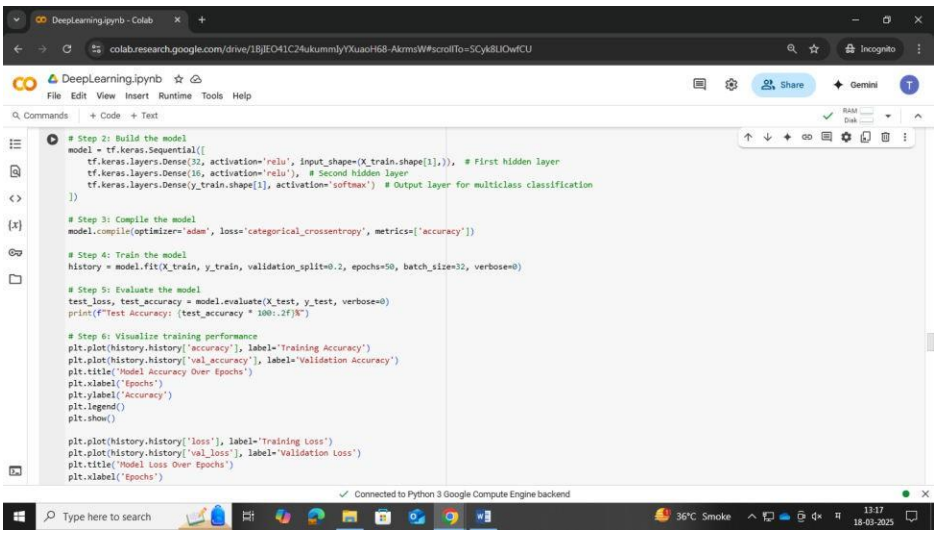
Code:

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt

# Step 1: Create a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=10, n_classes=3,
n_informative=8, random_state=42)
y = y.reshape(-1, 1)  # Reshape labels for encoding

# One-hot encode the labels
# One-hot encode the labels
encoder = OneHotEncoder(sparse_output=False)  # Use sparse_output instead
of sparse
y = encoder.fit_transform(y)

y = encoder.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 2: Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu',
input_shape=(X_train.shape[1],)),  # First hidden layer
    tf.keras.layers.Dense(16, activation='relu'),  # Second hidden layer
    tf.keras.layers.Dense(y_train.shape[1], activation='softmax')  #
Output layer for multiclass classification
])

# Step 3: Compile the model
```

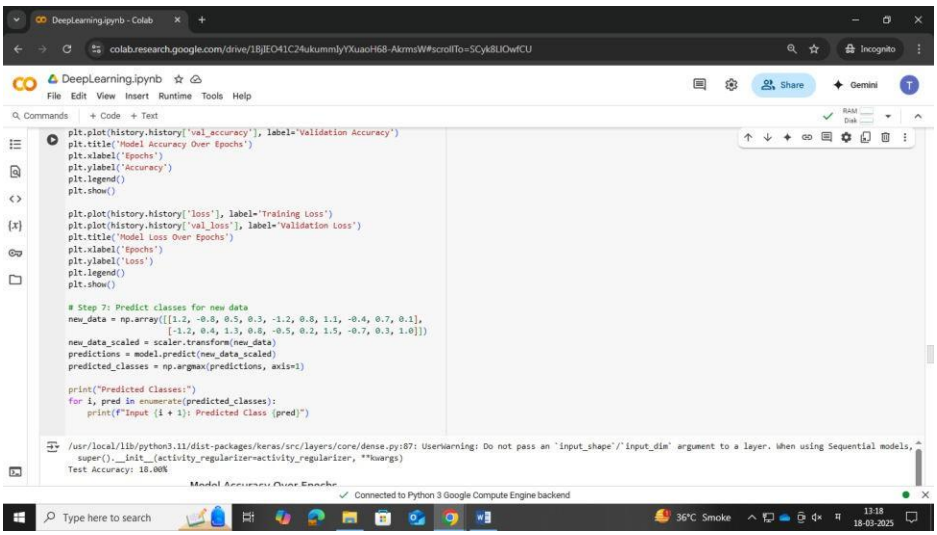
```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Step 4: Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, verbose=0)

# Step 5: Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

# Step 6: Visualize training performance
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 7: Predict classes for new data
new_data = np.array([[1.2, -0.8, 0.5, 0.3, -1.2, 0.8, 1.1, -0.4, 0.7,
0.1],
                     [-1.2, 0.4, 1.3, 0.8, -0.5, 0.2, 1.5, -0.7, 0.3,
1.0]])
new_data_scaled = scaler.transform(new_data)
predictions = model.predict(new_data_scaled)
predicted_classes = np.argmax(predictions, axis=1)

print("Predicted Classes:")
for i, pred in enumerate(predicted_classes):
    print(f"Input {i + 1}: Predicted Class {pred}")
```



```
import numpy as np
import tensorflow as tf
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt

# Step 1: Create a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=10, n_classes=3, n_informative=8, random_state=42)
y = y.reshape(-1, 1)  # Reshape labels for encoding

# One-hot encode the labels
# One-hot encode the labels
encoder = OneHotEncoder(sparse_output=False)  # Use sparse_output instead of sparse
y = encoder.fit_transform(y)

y = encoder.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 2: Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),  # First hidden layer
    tf.keras.layers.Dense(16, activation='relu'),  # Second hidden layer
```



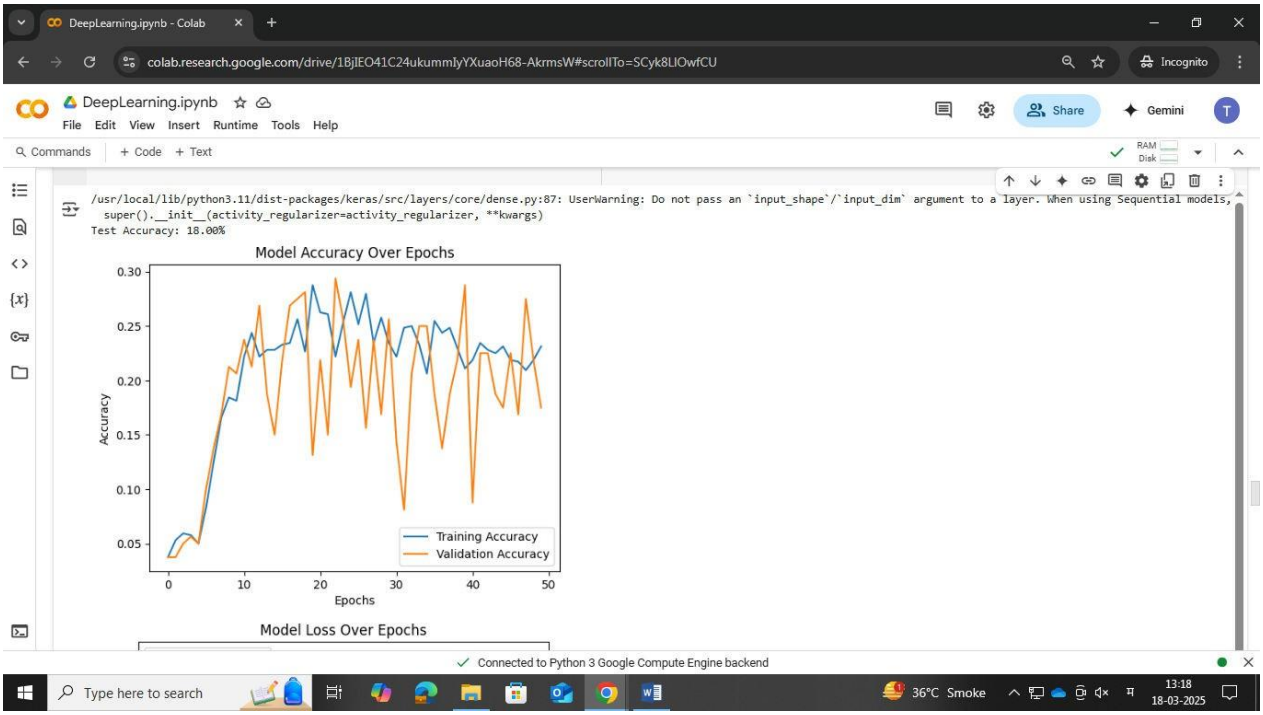
```
# Step 2: Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),  # First hidden layer
    tf.keras.layers.Dense(16, activation='relu'),  # Second hidden layer
    tf.keras.layers.Dense(y_train.shape[1], activation='softmax')  # Output layer for multiclass classification
])

# Step 3: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Step 4: Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=32, verbose=0)

# Step 5: Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

# Step 6: Visualize training performance
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epochs')
```

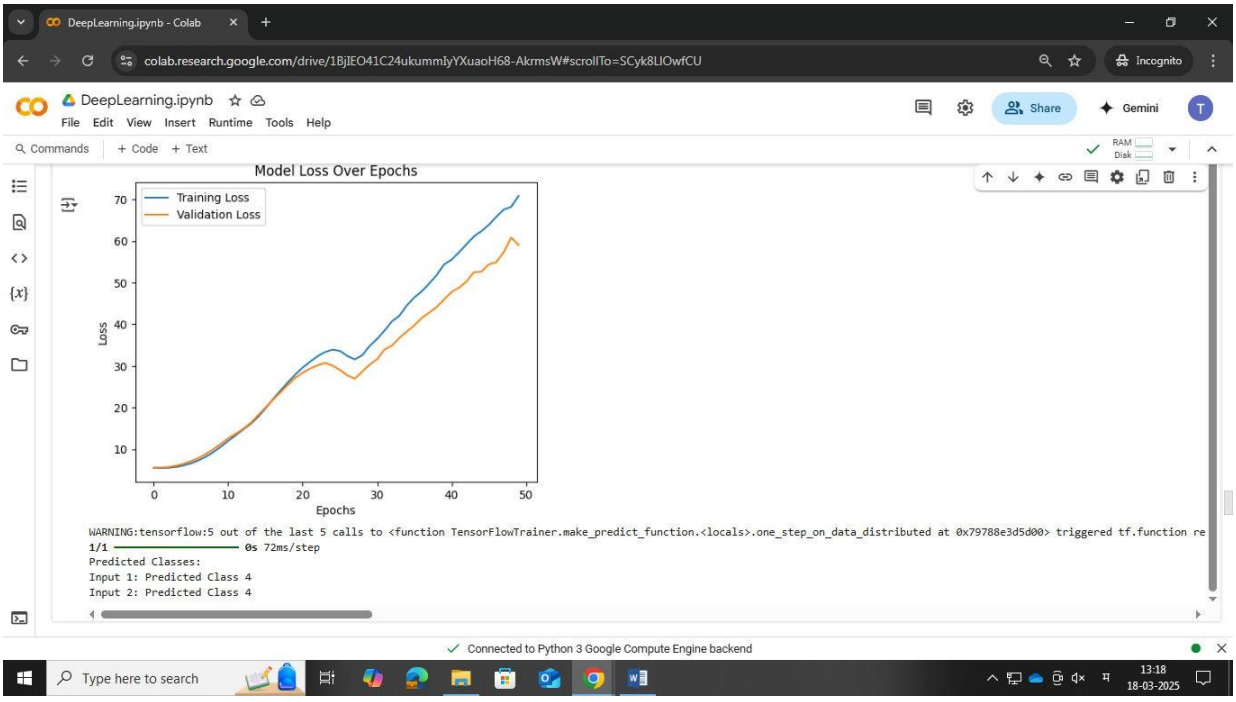


```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 7: Predict classes for new data
new_data = np.array([[1.2, -0.8, 0.5, 0.3, -1.2, 0.8, 1.1, -0.4, 0.7, 0.1],
                     [-1.2, 0.4, 1.3, 0.8, -0.5, 0.2, 1.5, -0.7, 0.3, 1.0]])
new_data_scaled = scaler.transform(new_data)
predictions = model.predict(new_data_scaled)
predicted_classes = np.argmax(predictions, axis=1)

print("Predicted Classes:")
for i, pred in enumerate(predicted_classes):
    print(f"Input {i + 1}: Predicted Class {pred}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Test Accuracy: 18.00%
```

Output:


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummIyYXuaoH68-AkrmsW#scrollTo=SCyk8LIOwfCU
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Test Accuracy: 18.00%
Model Accuracy Over Epochs
Accuracy
Epochs
Training Accuracy
Validation Accuracy
Model Loss Over Epochs
Connected to Python 3 Google Compute Engine backend
Type here to search
36°C Smoke
13:18
18-03-2025


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummIyYXuaoH68-AkrmsW#scrollTo=SCyk8LIOwfCU
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Model Loss Over Epochs
Training Loss
Validation Loss
Loss
Epochs
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x79788e3d5d00> triggered tf.function re
1/1 0s 72ms/step
Predicted Classes:
Input 1: Predicted Class 4
Input 2: Predicted Class 4
Connected to Python 3 Google Compute Engine backend
Type here to search
13:18
18-03-2025

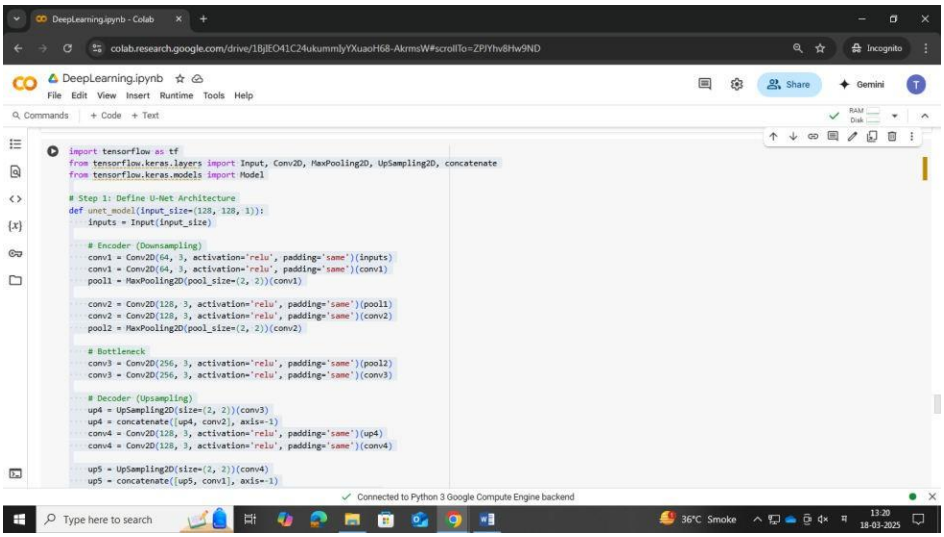
PRACTICAL 4.

Aim: Write a program to implement deep learning Techniques for image segmentation

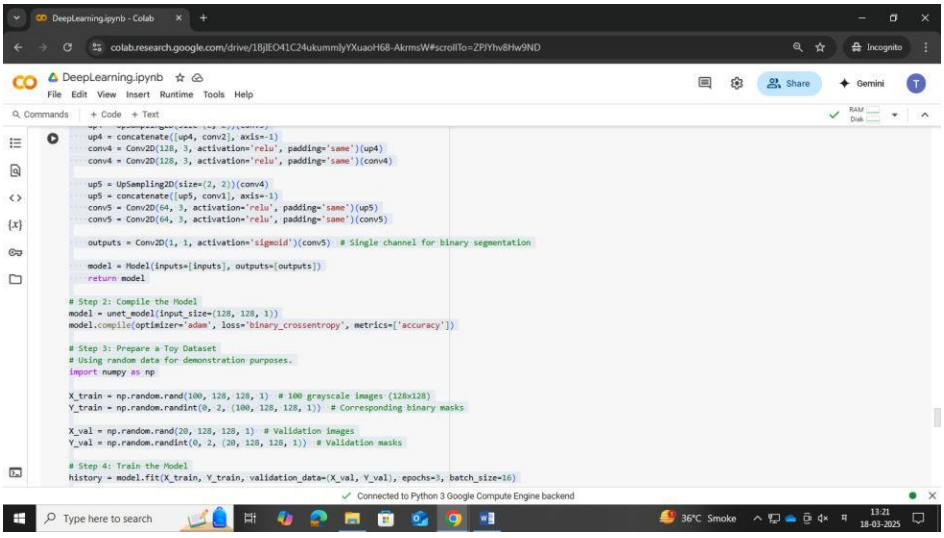
Code:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
UpSampling2D, concatenate
from tensorflow.keras.models import Model

# Step 1: Define U-Net Architecture
def unet_model(input_size=(128, 128, 1)):
    inputs = Input(input_size)

    # Encoder (Downsampling)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    # Bottleneck
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)

    # Decoder (Upsampling)
    up4 = UpSampling2D(size=(2, 2))(conv3)
    up4 = concatenate([up4, conv2], axis=-1)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(up4)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(conv4)

    up5 = UpSampling2D(size=(2, 2))(conv4)
    up5 = concatenate([up5, conv1], axis=-1)
    conv5 = Conv2D(64, 3, activation='relu', padding='same')(up5)
    conv5 = Conv2D(64, 3, activation='relu', padding='same')(conv5)

    outputs = Conv2D(1, 1, activation='sigmoid')(conv5)    # Single channel
for binary segmentation

    model = Model(inputs=[inputs], outputs=[outputs])
    return model
```

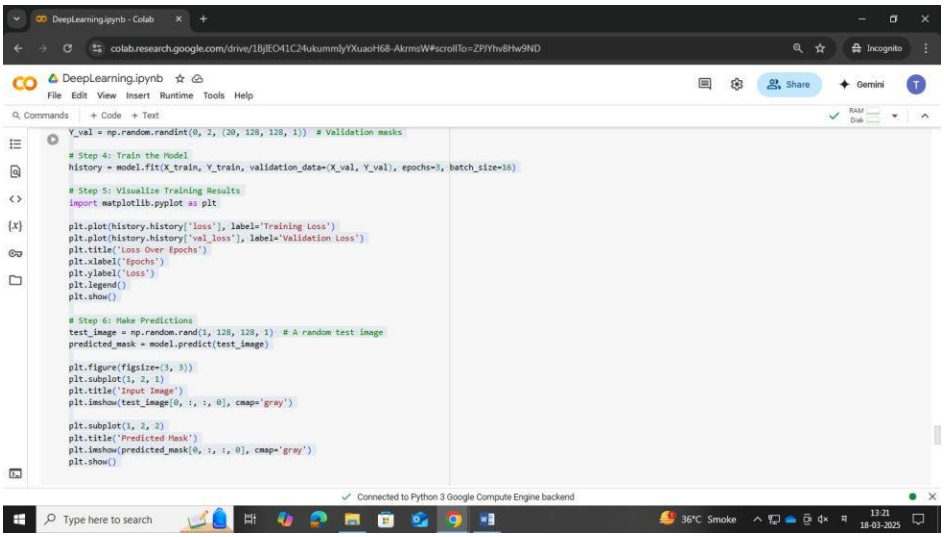
```
# Step 2: Compile the Model
model = unet_model(input_size=(128, 128, 1))
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Step 3: Prepare a Toy Dataset
# Using random data for demonstration purposes.
import numpy as np

X_train = np.random.rand(100, 128, 128, 1)    # 100 grayscale images
(128x128)
Y_train = np.random.randint(0, 2, (100, 128, 128, 1))    # Corresponding
binary masks

X_val = np.random.rand(20, 128, 128, 1)   # Validation images
Y_val = np.random.randint(0, 2, (20, 128, 128, 1))   # Validation masks

# Step 4: Train the Model
history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val),
epochs=3, batch_size=16)

# Step 5: Visualize Training Results
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
# Step 6: Make Predictions
test_image = np.random.rand(1, 128, 128, 1)    # A random test image
predicted_mask = model.predict(test_image)

plt.figure(figsize=(3, 3))
plt.subplot(1, 2, 1)
plt.title('Input Image')
plt.imshow(test_image[0, :, :, 0], cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Predicted Mask')
plt.imshow(predicted_mask[0, :, :, 0], cmap='gray')
plt.show()
```



```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate
from tensorflow.keras.models import Model

# Step 1: Define U-Net Architecture
def unet_model(input_size=(128, 128, 1)):
    inputs = Input(input_size)

    # Encoder (Downsampling)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    # Bottleneck
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)

    # Decoder (Upsampling)
    up4 = UpSampling2D(size=(2, 2))(conv3)
    up4 = concatenate([up4, conv2], axis=-1)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(up4)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(conv4)

    up5 = UpSampling2D(size=(2, 2))(conv4)
    up5 = concatenate([up5, conv1], axis=-1)
```

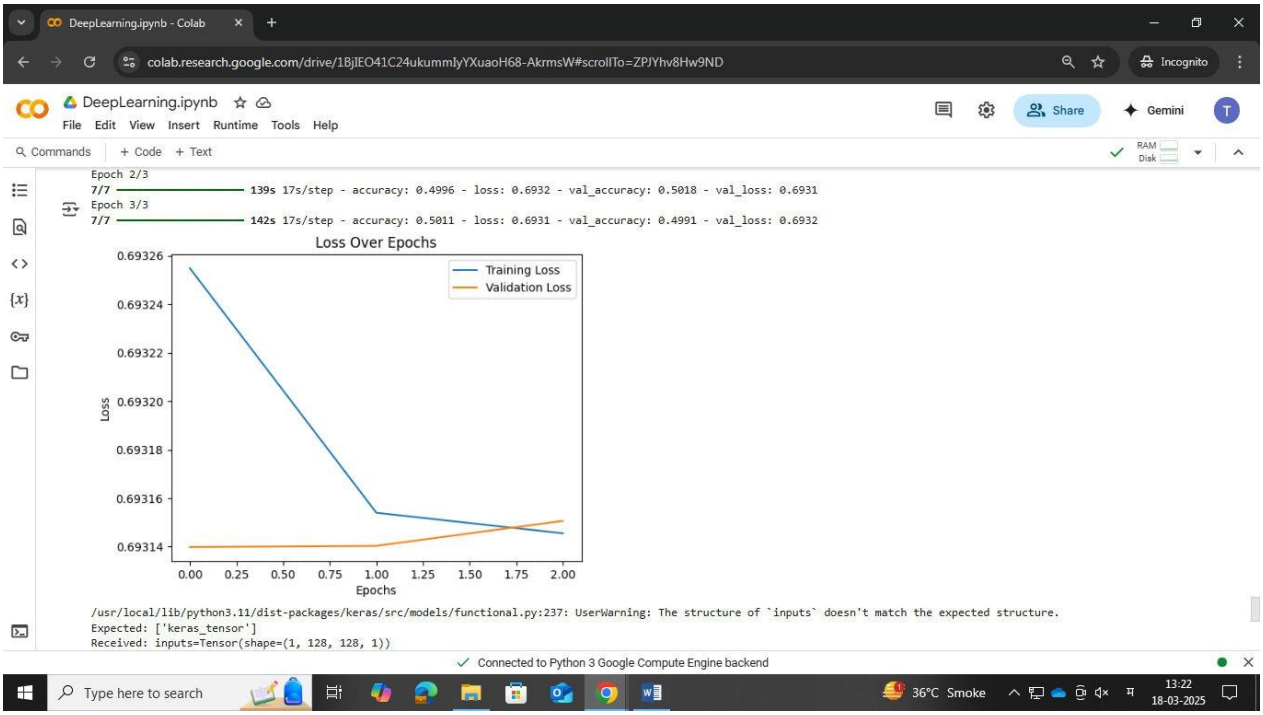


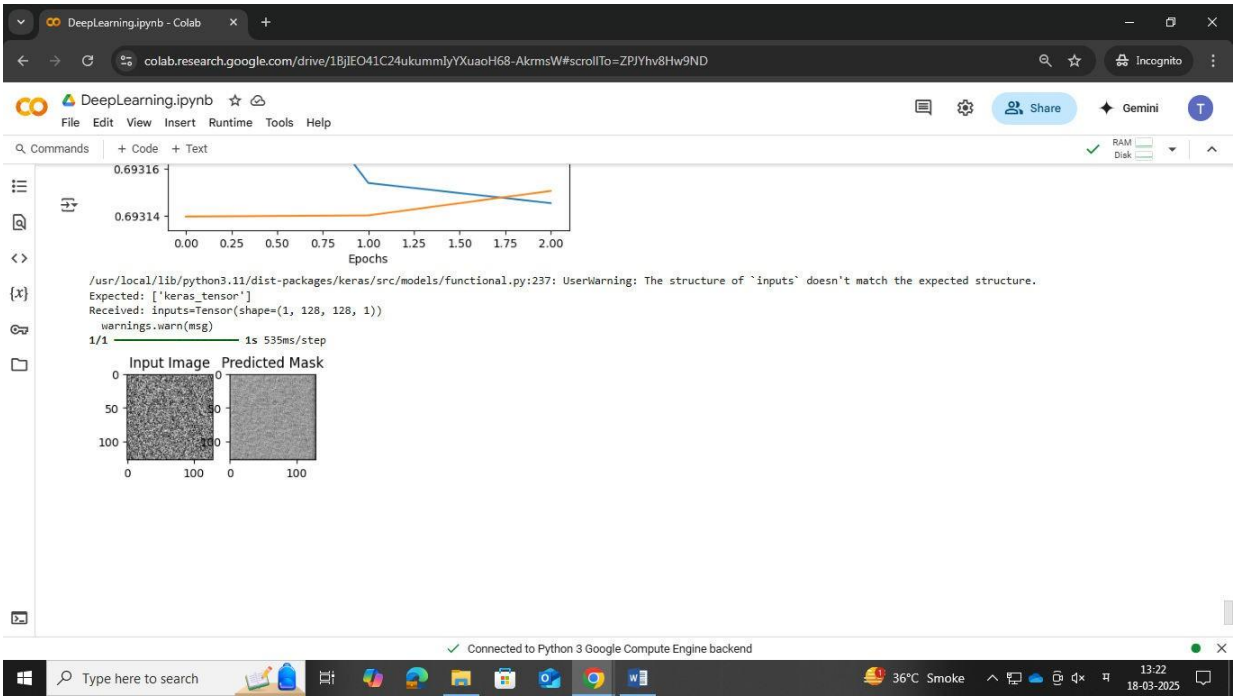
```
    up4 = concatenate([up4, conv2], axis=-1)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(up4)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(conv4)

    up5 = UpSampling2D(size=(2, 2))(conv4)
    up5 = concatenate([up5, conv1], axis=-1)
    conv5 = Conv2D(64, 3, activation='relu', padding='same')(up5)
    conv5 = Conv2D(64, 3, activation='relu', padding='same')(conv5)

    outputs = Conv2D(1, 1, activation='sigmoid')(conv5)  # Single channel for binary segmentation

    model = Model(inputs=[inputs], outputs=[outputs])
    return model

# Step 2: Compile the Model
model = unet_model(input_size=(128, 128, 1))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 3: Prepare a Toy Dataset
# Using random data for demonstration purposes.
import numpy as np

X_train = np.random.rand(100, 128, 128, 1)  # 100 grayscale images (128x128)
Y_train = np.random.randint(0, 2, (100, 128, 128, 1))  # Corresponding binary masks

X_val = np.random.rand(20, 128, 128, 1)  # Validation images
Y_val = np.random.randint(0, 2, (20, 128, 128, 1))  # Validation masks

# Step 4: Train the Model
history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=3, batch_size=16)
```



```
Y_val = np.random.randint(0, 2, (20, 128, 128, 1))  # Validation masks

# Step 4: Train the Model
history = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=3, batch_size=16)

# Step 5: Visualize Training Results
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 6: Make Predictions
test_image = np.random.rand(1, 128, 128, 1)  # A random test image
predicted_mask = model.predict(test_image)

plt.figure(figsize=(3, 3))
plt.subplot(1, 2, 1)
plt.title('Input Image')
plt.imshow(test_image[0, :, :, 0], cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Predicted Mask')
plt.imshow(predicted_mask[0, :, :, 0], cmap='gray')
plt.show()
```

Output:


DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummIyYXuaoH68-AkrmsW#scrollTo=ZPJYhv8Hw9ND
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Epoch 2/3
7/7 139s 17s/step - accuracy: 0.4996 - loss: 0.6932 - val_accuracy: 0.5018 - val_loss: 0.6931
Epoch 3/3
7/7 142s 17s/step - accuracy: 0.5011 - loss: 0.6931 - val_accuracy: 0.4991 - val_loss: 0.6932
Loss Over Epochs
Training Loss
Validation Loss
Loss
Epochs
/usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The structure of `inputs` doesn't match the expected structure.
Expected: ['keras_tensor']
Received: inputs=Tensor(shape=(1, 128, 128, 1))
Connected to Python 3 Google Compute Engine backend
Type here to search
36°C Smoke
13:22
18-03-2025

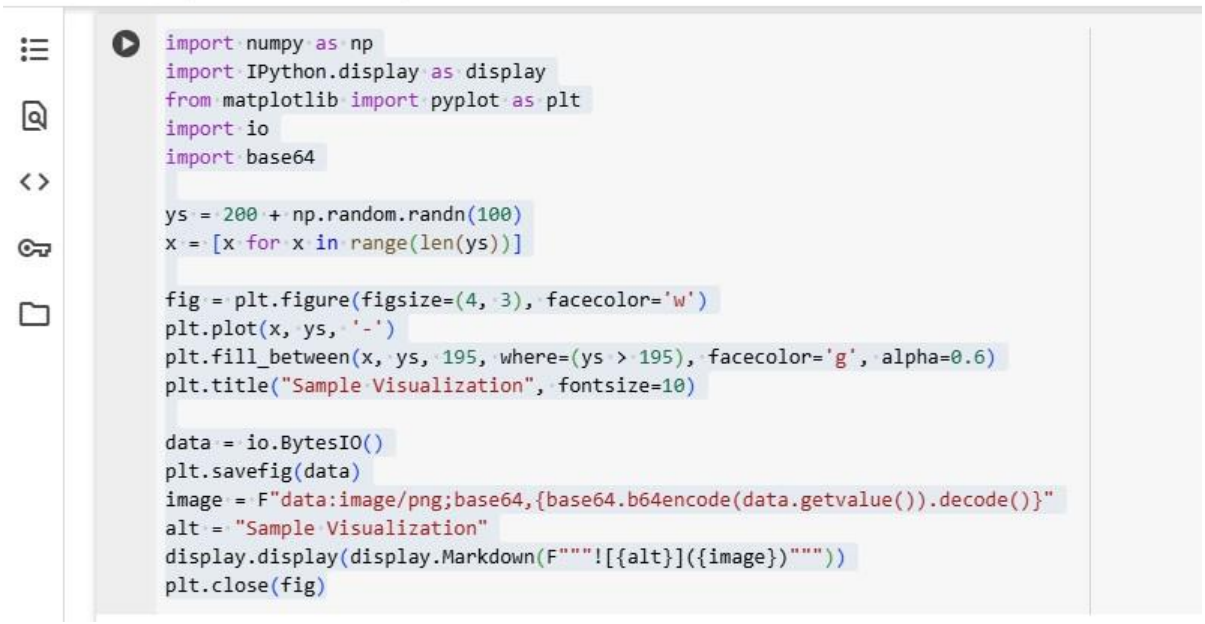

DeepLearning.ipynb - Colab
colab.research.google.com/drive/1BjIEO41C24ukummIyYXuaoH68-AkrmsW#scrollTo=ZPJYhv8Hw9ND
DeepLearning.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Epochs
/usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The structure of `inputs` doesn't match the expected structure.
Expected: ['keras_tensor']
Received: inputs=Tensor(shape=(1, 128, 128, 1))
warnings.warn(msg)
1/1 1s 535ms/step
Input Image
Predicted Mask
Connected to Python 3 Google Compute Engine backend
Type here to search
36°C Smoke
13:22
18-03-2025

PRACTICAL 5.

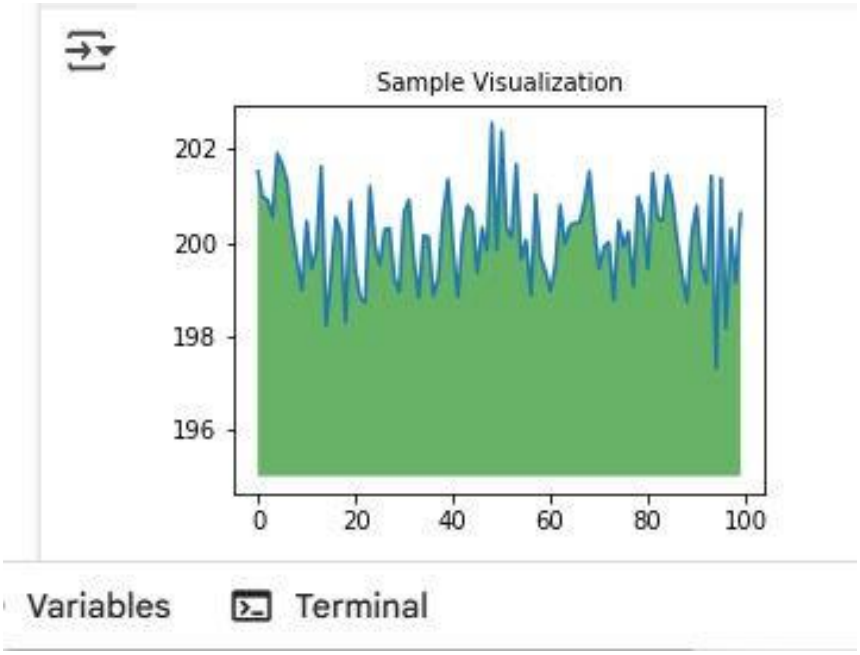
Aim: Write a program to predict a caption for a sample image using LSTM.

Code:

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image =
F"{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
```



Output:


Sample Visualization
202
200
198
196
0
20
40
60
80
100
Variables
Terminal

PRACTICAL 6.

Aim: Applying the Autoencoder algorithms for encoding real-world data

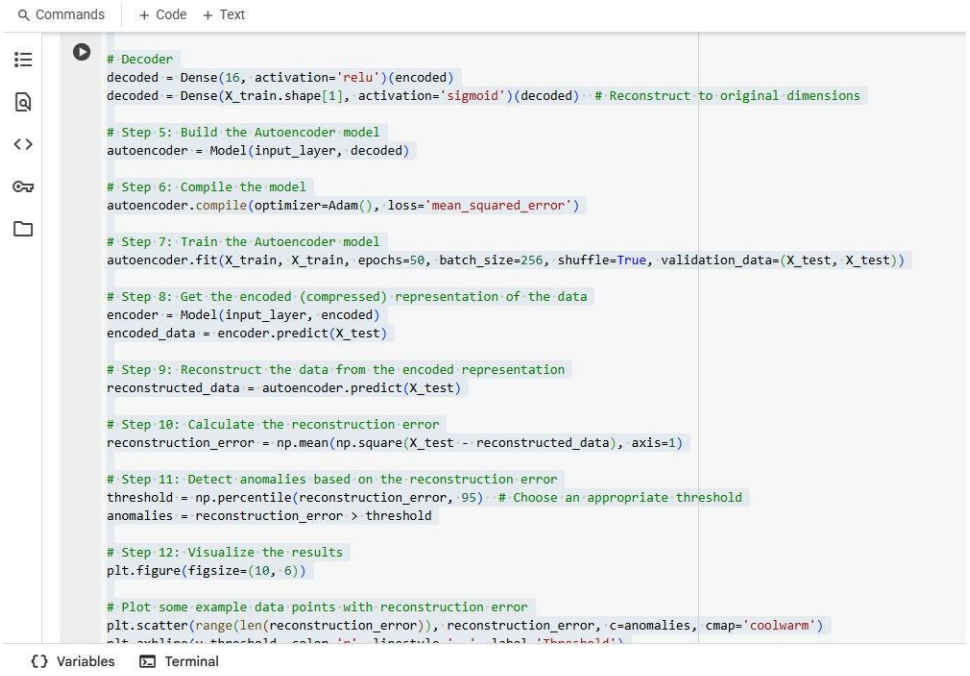
Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Model
from keras.layers import Input, Dense
from keras.optimizers import Adam
from keras import regularizers

# Step 1: Generate or load the dataset (here, we're using synthetic data)
# Example: 1000 samples, 20 features (could represent customer data,
financial data, etc.)
np.random.seed(42)
data = np.random.rand(1000, 20)

# Step 2: Normalize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Step 3: Split data into training and test sets
X_train, X_test = train_test_split(data_scaled, test_size=0.2,
random_state=42)

# Step 4: Define the Autoencoder architecture
input_layer = Input(shape=(X_train.shape[1],))

# Encoder
encoded = Dense(16, activation='relu',
activity_regularizer=regularizers.l2(0.01))(input_layer)
encoded = Dense(8, activation='relu')(encoded)    # Compress the data to 8
features

# Decoder
decoded = Dense(16, activation='relu')(encoded)
decoded = Dense(X_train.shape[1], activation='sigmoid')(decoded)    #
Reconstruct to original dimensions

# Step 5: Build the Autoencoder model
```

```
autoencoder = Model(input_layer, decoded)

# Step 6: Compile the model
autoencoder.compile(optimizer=Adam(), loss='mean_squared_error')

# Step 7: Train the Autoencoder model
autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True,
validation_data=(X_test, X_test))

# Step 8: Get the encoded (compressed) representation of the data
encoder = Model(input_layer, encoded)
encoded_data = encoder.predict(X_test)

# Step 9: Reconstruct the data from the encoded representation
reconstructed_data = autoencoder.predict(X_test)

# Step 10: Calculate the reconstruction error
reconstruction_error = np.mean(np.square(X_test - reconstructed_data),
axis=1)

# Step 11: Detect anomalies based on the reconstruction error
threshold = np.percentile(reconstruction_error, 95)    # Choose an
appropriate threshold
anomalies = reconstruction_error > threshold

# Step 12: Visualize the results
plt.figure(figsize=(10, 6))

# Plot some example data points with reconstruction error
plt.scatter(range(len(reconstruction_error)), reconstruction_error,
c=anomalies, cmap='coolwarm')
plt.axhline(y=threshold, color='r', linestyle='--', label='Threshold')
plt.title('Reconstruction Error with Anomalies Detected')
plt.xlabel('Sample Index')
plt.ylabel('Reconstruction Error')
plt.legend()
plt.show()

# Optionally, you can print out the indices of detected anomalies
print("Anomalies detected at indices:", np.where(anomalies)[0])
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Model
from keras.layers import Input, Dense
from keras.optimizers import Adam
from keras import regularizers

# Step 1: Generate or load the dataset (here, we're using synthetic data)
# Example: 1000 samples, 20 features (could represent customer data, financial data, etc.)
np.random.seed(42)
data = np.random.rand(1000, 20)

# Step 2: Normalize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Step 3: Split data into training and test sets
X_train, X_test = train_test_split(data_scaled, test_size=0.2, random_state=42)

# Step 4: Define the Autoencoder architecture
input_layer = Input(shape=(X_train.shape[1],))

# Encoder
encoded = Dense(16, activation='relu', activity_regularizer=regularizers.l2(0.01))(input_layer)
encoded = Dense(8, activation='relu')(encoded)  # Compress the data to 8 features

# Decoder
decoded = Dense(16, activation='relu')(encoded)
decoded = Dense(X_train.shape[1], activation='sigmoid')(decoded)  # Reconstruct to original dimensions
```

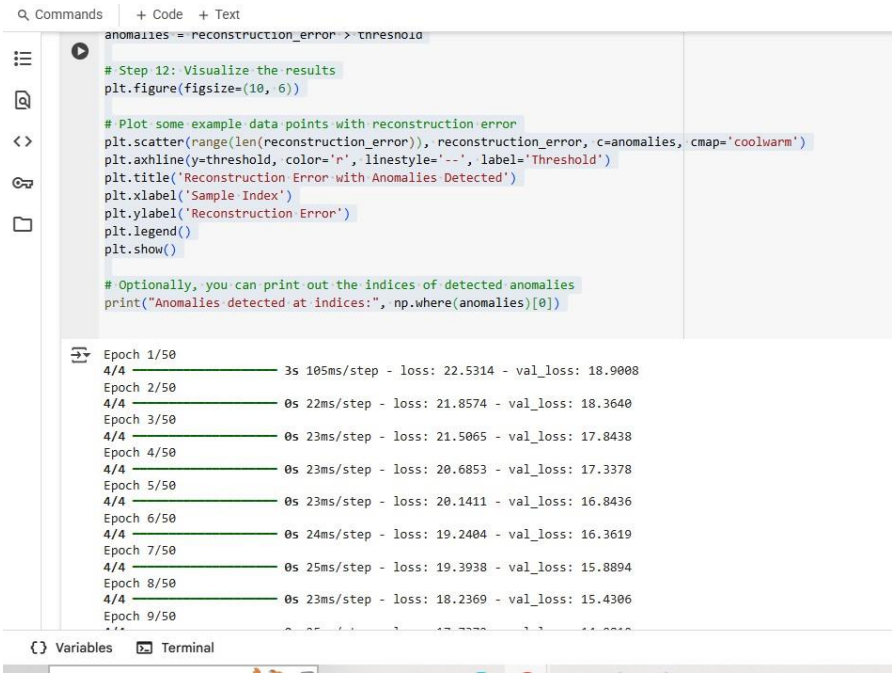
Variables Terminal

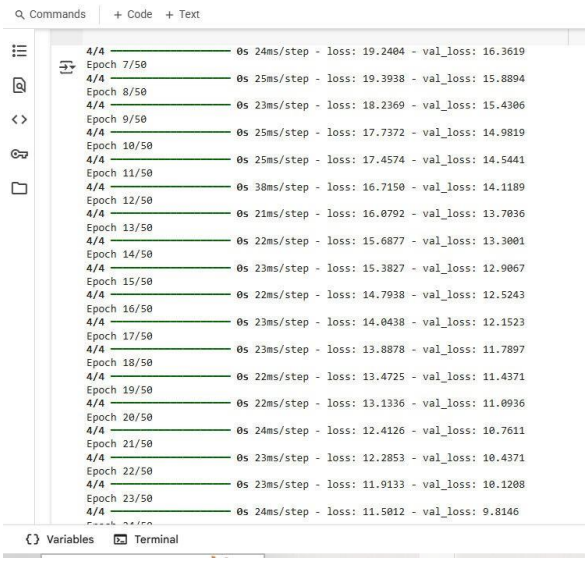


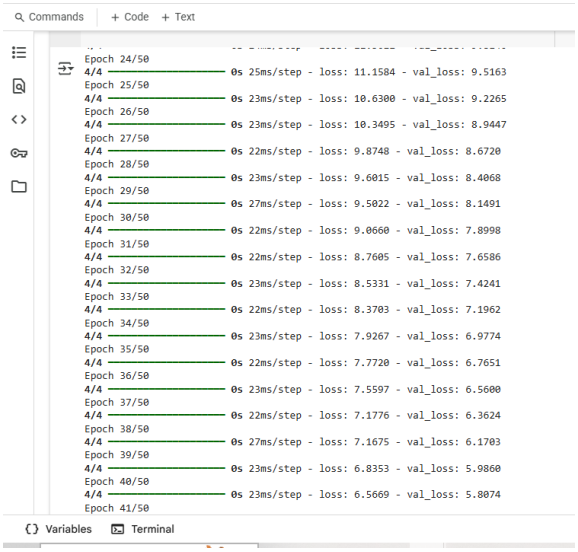
Commands + Code + Text

```
# Decoder
decoded = Dense(16, activation='relu')(encoded)
decoded = Dense(X_train.shape[1], activation='sigmoid')(decoded)  # Reconstruct to original dimensions

# Step 5: Build the Autoencoder model
autoencoder = Model(input_layer, decoded)

# Step 6: Compile the model
autoencoder.compile(optimizer=Adam(), loss='mean_squared_error')

# Step 7: Train the Autoencoder model
autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True, validation_data=(X_test, X_test))

# Step 8: Get the encoded (compressed) representation of the data
encoder = Model(input_layer, encoded)
encoded_data = encoder.predict(X_test)

# Step 9: Reconstruct the data from the encoded representation
reconstructed_data = autoencoder.predict(X_test)

# Step 10: Calculate the reconstruction error
reconstruction_error = np.mean(np.square(X_test - reconstructed_data), axis=1)

# Step 11: Detect anomalies based on the reconstruction error
threshold = np.percentile(reconstruction_error, 95)  # Choose an appropriate threshold
anomalies = reconstruction_error > threshold

# Step 12: Visualize the results
plt.figure(figsize=(10, 6))

# Plot some example data points with reconstruction error
plt.scatter(range(len(reconstruction_error)), reconstruction_error, c=anomalies, cmap='coolwarm')
```

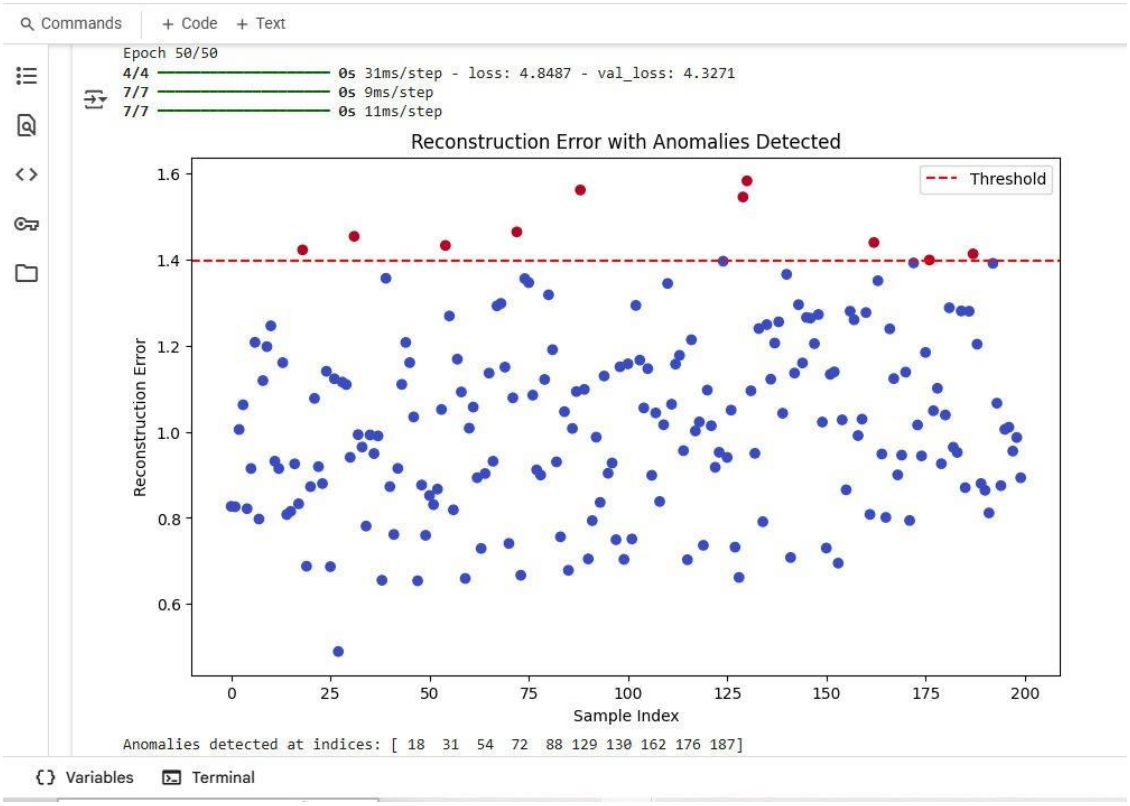
Variables Terminal


Commands
+ Code
+ Text
anomalies = reconstruction_error > threshold
# Step 12: Visualize the results
plt.figure(figsize=(10, 6))
# Plot some example data points with reconstruction error
plt.scatter(range(len(reconstruction_error)), reconstruction_error, c=anomalies, cmap='coolwarm')
plt.axhline(y=threshold, color='r', linestyle='--', label='Threshold')
plt.title('Reconstruction Error with Anomalies Detected')
plt.xlabel('Sample Index')
plt.ylabel('Reconstruction Error')
plt.legend()
plt.show()
# Optionally, you can print out the indices of detected anomalies
print("Anomalies detected at indices:", np.where(anomalies)[0])
Epoch 1/50
4/4 3s 105ms/step - loss: 22.5314 - val_loss: 18.9008
Epoch 2/50
4/4 0s 22ms/step - loss: 21.8574 - val_loss: 18.3640
Epoch 3/50
4/4 0s 23ms/step - loss: 21.5065 - val_loss: 17.8438
Epoch 4/50
4/4 0s 23ms/step - loss: 20.6853 - val_loss: 17.3378
Epoch 5/50
4/4 0s 23ms/step - loss: 20.1411 - val_loss: 16.8436
Epoch 6/50
4/4 0s 24ms/step - loss: 19.2404 - val_loss: 16.3619
Epoch 7/50
4/4 0s 25ms/step - loss: 19.3938 - val_loss: 15.8894
Epoch 8/50
4/4 0s 23ms/step - loss: 18.2369 - val_loss: 15.4306
Epoch 9/50
Variables
Terminal


Commands
+ Code
+ Text
4/4 0s 24ms/step - loss: 19.2404 - val_loss: 16.3619
Epoch 7/50
4/4 0s 25ms/step - loss: 19.3938 - val_loss: 15.8894
Epoch 8/50
4/4 0s 23ms/step - loss: 18.2369 - val_loss: 15.4306
Epoch 9/50
4/4 0s 25ms/step - loss: 17.7372 - val_loss: 14.9819
Epoch 10/50
4/4 0s 25ms/step - loss: 17.4574 - val_loss: 14.5441
Epoch 11/50
4/4 0s 38ms/step - loss: 16.7150 - val_loss: 14.1189
Epoch 12/50
4/4 0s 21ms/step - loss: 16.0792 - val_loss: 13.7036
Epoch 13/50
4/4 0s 22ms/step - loss: 15.6877 - val_loss: 13.3001
Epoch 14/50
4/4 0s 23ms/step - loss: 15.3827 - val_loss: 12.9067
Epoch 15/50
4/4 0s 22ms/step - loss: 14.7938 - val_loss: 12.5243
Epoch 16/50
4/4 0s 23ms/step - loss: 14.0438 - val_loss: 12.1523
Epoch 17/50
4/4 0s 23ms/step - loss: 13.8878 - val_loss: 11.7897
Epoch 18/50
4/4 0s 22ms/step - loss: 13.4725 - val_loss: 11.4371
Epoch 19/50
4/4 0s 22ms/step - loss: 13.1336 - val_loss: 11.0936
Epoch 20/50
4/4 0s 24ms/step - loss: 12.4126 - val_loss: 10.7611
Epoch 21/50
4/4 0s 23ms/step - loss: 12.2853 - val_loss: 10.4371
Epoch 22/50
4/4 0s 23ms/step - loss: 11.9133 - val_loss: 10.1208
Epoch 23/50
4/4 0s 24ms/step - loss: 11.5012 - val_loss: 9.8146
Variables
Terminal


Commands
+ Code
+ Text
Epoch 24/50
4/4 0s 25ms/step - loss: 11.1584 - val_loss: 9.5163
Epoch 25/50
4/4 0s 23ms/step - loss: 10.6300 - val_loss: 9.2265
Epoch 26/50
4/4 0s 23ms/step - loss: 10.3495 - val_loss: 8.9447
Epoch 27/50
4/4 0s 22ms/step - loss: 9.8748 - val_loss: 8.6720
Epoch 28/50
4/4 0s 23ms/step - loss: 9.6015 - val_loss: 8.4068
Epoch 29/50
4/4 0s 27ms/step - loss: 9.5022 - val_loss: 8.1491
Epoch 30/50
4/4 0s 22ms/step - loss: 9.0660 - val_loss: 7.8998
Epoch 31/50
4/4 0s 22ms/step - loss: 8.7605 - val_loss: 7.6586
Epoch 32/50
4/4 0s 23ms/step - loss: 8.5331 - val_loss: 7.4241
Epoch 33/50
4/4 0s 22ms/step - loss: 8.3703 - val_loss: 7.1962
Epoch 34/50
4/4 0s 23ms/step - loss: 7.9267 - val_loss: 6.9774
Epoch 35/50
4/4 0s 22ms/step - loss: 7.7720 - val_loss: 6.7651
Epoch 36/50
4/4 0s 23ms/step - loss: 7.5597 - val_loss: 6.5600
Epoch 37/50
4/4 0s 22ms/step - loss: 7.1776 - val_loss: 6.3624
Epoch 38/50
4/4 0s 27ms/step - loss: 7.1675 - val_loss: 6.1703
Epoch 39/50
4/4 0s 23ms/step - loss: 6.8353 - val_loss: 5.9860
Epoch 40/50
4/4 0s 23ms/step - loss: 6.5669 - val_loss: 5.8074
Epoch 41/50
Variables
Terminal

Output:


Commands
+ Code
+ Text
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 4.8487 - val_loss: 4.3271
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
Reconstruction Error with Anomalies Detected
Threshold
Reconstruction Error
Sample Index
1.6
1.4
1.2
1.0
0.8
0.6
0
25
50
75
100
125
150
175
200
Anomalies detected at indices: [18 31 54 72 88 129 130 162 176 187]
Variables
Terminal

PRACTICAL 7.

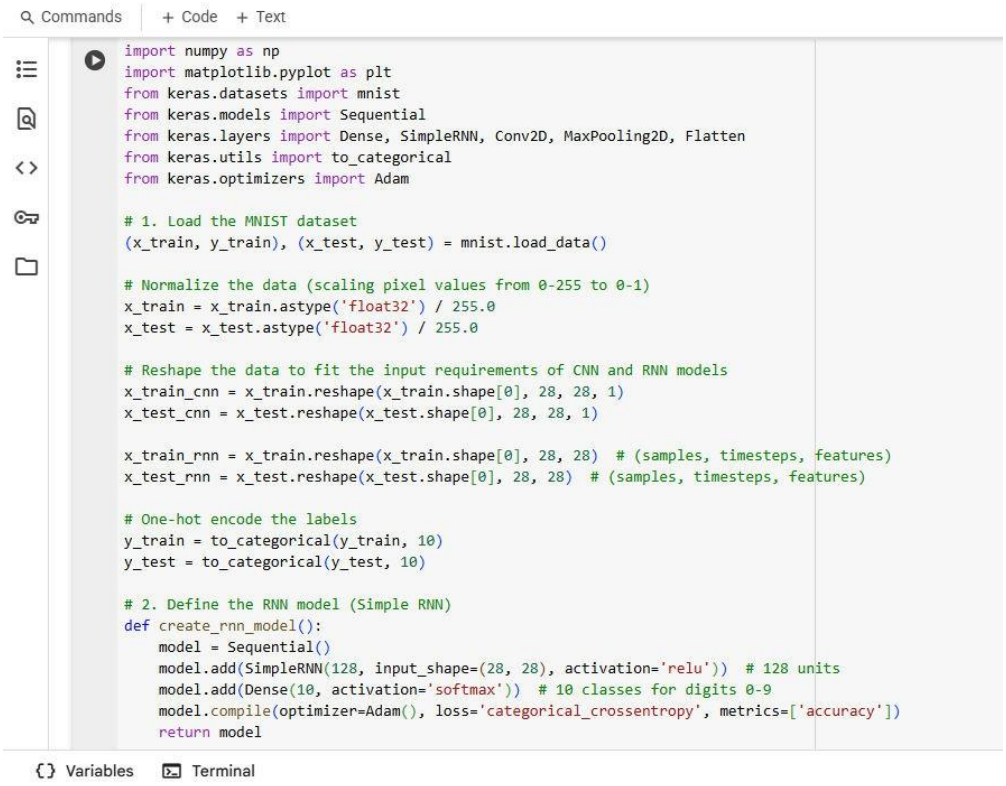
Aim: Write a program for character recognition using RNN and compare it with CNN.

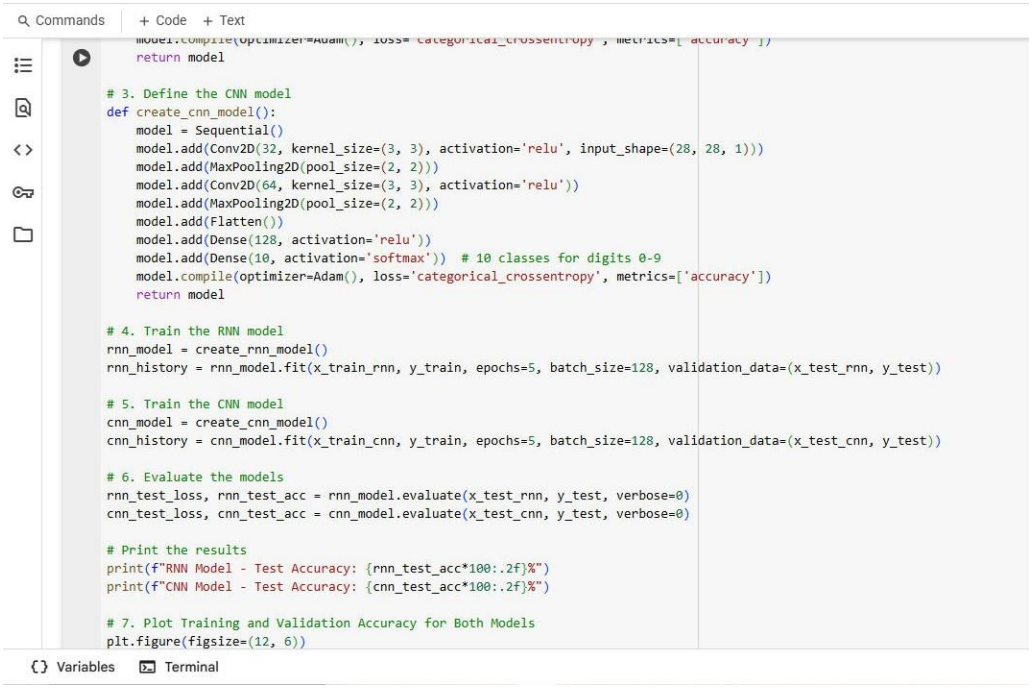
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Conv2D, MaxPooling2D, Flatten
from keras.utils import to_categorical
from keras.optimizers import Adam

# 1. Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the data (scaling pixel values from 0-255 to 0-1)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape the data to fit the input requirements of CNN and RNN models
x_train_cnn = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test_cnn = x_test.reshape(x_test.shape[0], 28, 28, 1)

x_train_rnn = x_train.reshape(x_train.shape[0], 28, 28)    # (samples,
timesteps, features)
x_test_rnn = x_test.reshape(x_test.shape[0], 28, 28)    # (samples,
timesteps, features)

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 2. Define the RNN model (Simple RNN)
def create_rnn_model():
    model = Sequential()
    model.add(SimpleRNN(128, input_shape=(28, 28), activation='relu'))   #
128 units
    model.add(Dense(10, activation='softmax'))   # 10 classes for digits 0-
9
    model.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
```

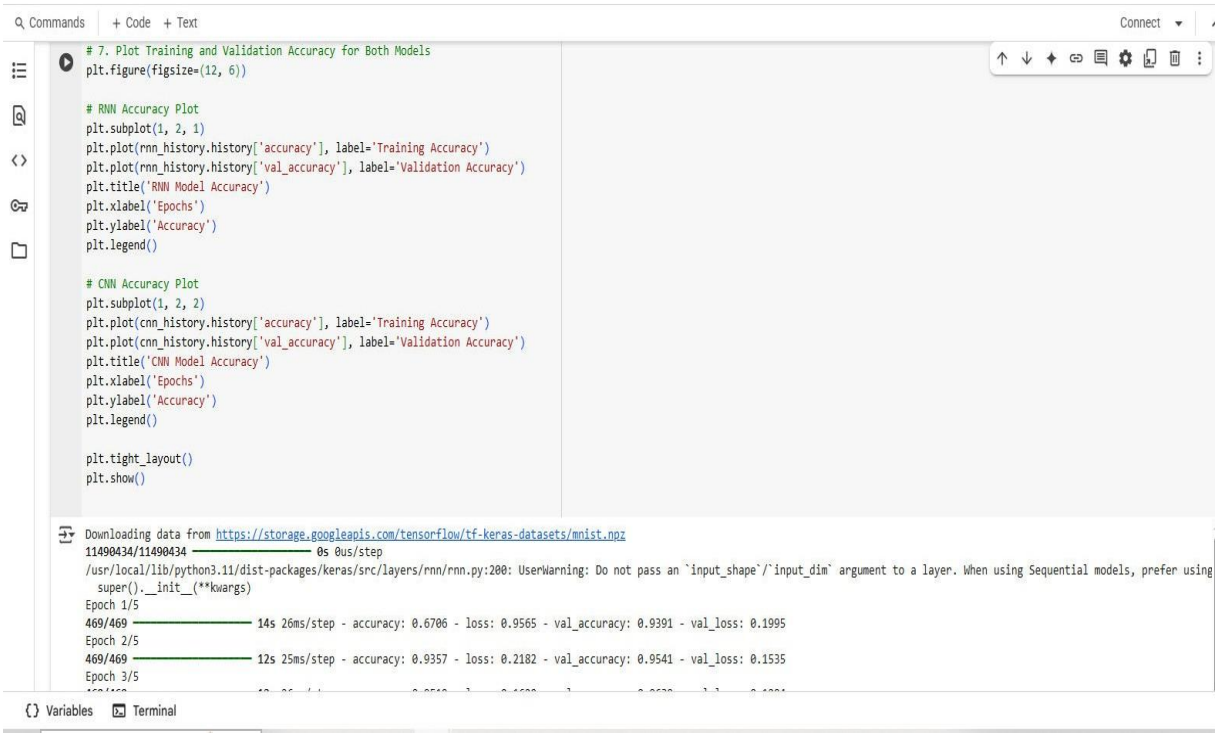
```
# 3. Define the CNN model
def create_cnn_model():
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))   # 10 classes for digits 0-
9
    model.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

# 4. Train the RNN model
rnn_model = create_rnn_model()
rnn_history = rnn_model.fit(x_train_rnn, y_train, epochs=5,
batch_size=128, validation_data=(x_test_rnn, y_test))

# 5. Train the CNN model
cnn_model = create_cnn_model()
cnn_history = cnn_model.fit(x_train_cnn, y_train, epochs=5,
batch_size=128, validation_data=(x_test_cnn, y_test))

# 6. Evaluate the models
rnn_test_loss, rnn_test_acc = rnn_model.evaluate(x_test_rnn, y_test,
verbose=0)
cnn_test_loss, cnn_test_acc = cnn_model.evaluate(x_test_cnn, y_test,
verbose=0)

# Print the results
print(f"RNN Model - Test Accuracy: {rnn_test_acc*100:.2f}%")
print(f"CNN Model - Test Accuracy: {cnn_test_acc*100:.2f}%")

# 7. Plot Training and Validation Accuracy for Both Models
plt.figure(figsize=(12, 6))

# RNN Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(rnn_history.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('RNN Model Accuracy')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# CNN Accuracy Plot
plt.subplot(1, 2, 2)
plt.plot(cnn_history.history['accuracy'], label='Training Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```





```
    return model

# 3. Define the CNN model
def create_cnn_model():
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))  # 10 classes for digits 0-9
    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# 4. Train the RNN model
rnn_model = create_rnn_model()
rnn_history = rnn_model.fit(x_train_rnn, y_train, epochs=5, batch_size=128, validation_data=(x_test_rnn, y_test))

# 5. Train the CNN model
cnn_model = create_cnn_model()
cnn_history = cnn_model.fit(x_train_cnn, y_train, epochs=5, batch_size=128, validation_data=(x_test_cnn, y_test))

# 6. Evaluate the models
rnn_test_loss, rnn_test_acc = rnn_model.evaluate(x_test_rnn, y_test, verbose=0)
cnn_test_loss, cnn_test_acc = cnn_model.evaluate(x_test_cnn, y_test, verbose=0)

# Print the results
print(f"RNN Model - Test Accuracy: {rnn_test_acc*100:.2f}%")
print(f"CNN Model - Test Accuracy: {cnn_test_acc*100:.2f}%")

# 7. Plot Training and Validation Accuracy for Both Models
plt.figure(figsize=(12, 6))
```

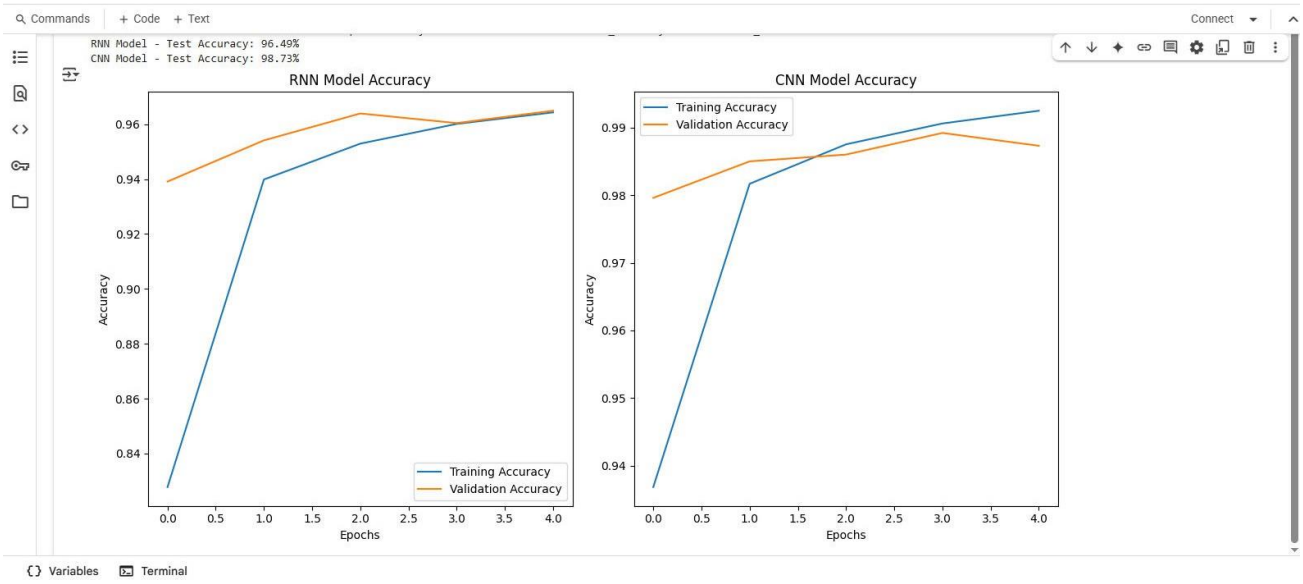


```
# 7. Plot Training and Validation Accuracy for Both Models
plt.figure(figsize=(12, 6))

# RNN Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(rnn_history.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('RNN Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# CNN Accuracy Plot
plt.subplot(1, 2, 2)
plt.plot(cnn_history.history['accuracy'], label='Training Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using
  super().__init__(**kwargs)
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.6706 - loss: 0.9565 - val_accuracy: 0.9391 - val_loss: 0.1995
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 12s 25ms/step - accuracy: 0.9357 - loss: 0.2182 - val_accuracy: 0.9541 - val_loss: 0.1535
Epoch 3/5
```


Commands
+ Code
+ Text
Connect
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using
super().__init__(**kwargs)
Epoch 1/5
469/469 14s 26ms/step - accuracy: 0.6706 - loss: 0.9565 - val_accuracy: 0.9391 - val_loss: 0.1995
Epoch 2/5
469/469 12s 25ms/step - accuracy: 0.9357 - loss: 0.2182 - val_accuracy: 0.9541 - val_loss: 0.1535
Epoch 3/5
469/469 12s 26ms/step - accuracy: 0.9519 - loss: 0.1620 - val_accuracy: 0.9639 - val_loss: 0.1294
Epoch 4/5
469/469 21s 27ms/step - accuracy: 0.9609 - loss: 0.1351 - val_accuracy: 0.9604 - val_loss: 0.1275
Epoch 5/5
469/469 20s 26ms/step - accuracy: 0.9653 - loss: 0.1177 - val_accuracy: 0.9649 - val_loss: 0.1252
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mode
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
469/469 46s 95ms/step - accuracy: 0.8515 - loss: 0.4867 - val_accuracy: 0.9796 - val_loss: 0.0672
Epoch 2/5
469/469 44s 94ms/step - accuracy: 0.9806 - loss: 0.0632 - val_accuracy: 0.9850 - val_loss: 0.0455
Epoch 3/5
469/469 44s 93ms/step - accuracy: 0.9872 - loss: 0.0397 - val_accuracy: 0.9860 - val_loss: 0.0371
Epoch 4/5
469/469 43s 92ms/step - accuracy: 0.9903 - loss: 0.0309 - val_accuracy: 0.9892 - val_loss: 0.0334
Epoch 5/5
469/469 83s 94ms/step - accuracy: 0.9930 - loss: 0.0230 - val_accuracy: 0.9873 - val_loss: 0.0374
RNN Model - Test Accuracy: 96.49%
CNN Model - Test Accuracy: 98.73%
RNN Model Accuracy
CNN Model Accuracy
Training Accuracy
Validation Accuracy
Variables
Terminal

Output:


Commands
+ Code
+ Text
Connect
RNN Model - Test Accuracy: 96.49%
CNN Model - Test Accuracy: 98.73%
RNN Model Accuracy
CNN Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
Variables
Terminal

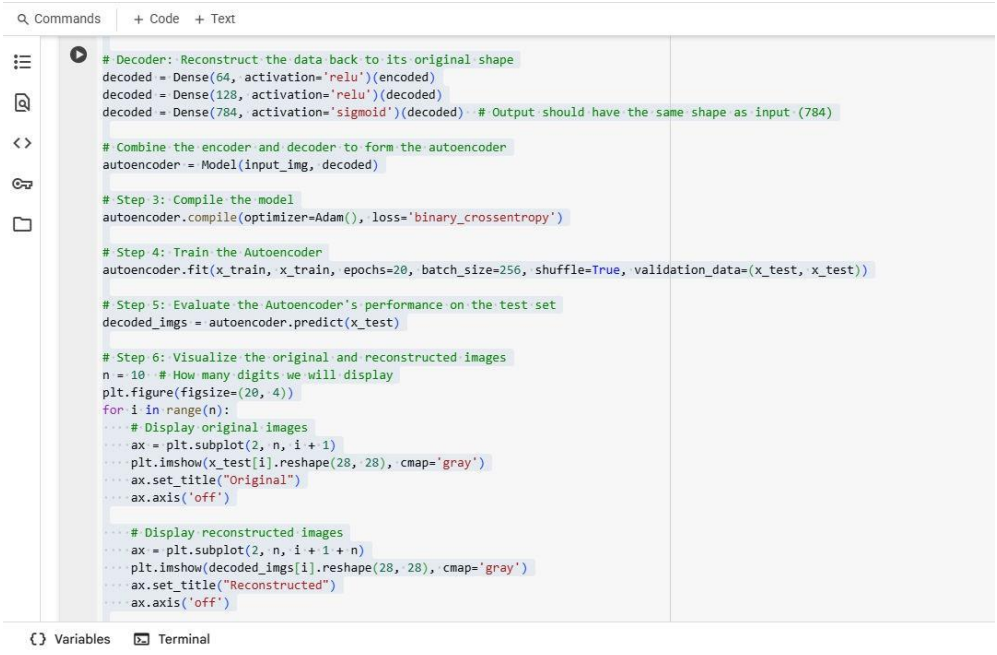
PRACTICAL 8.

Aim: Write a program to develop Autoencoders using MNIST Handwritten Digits.

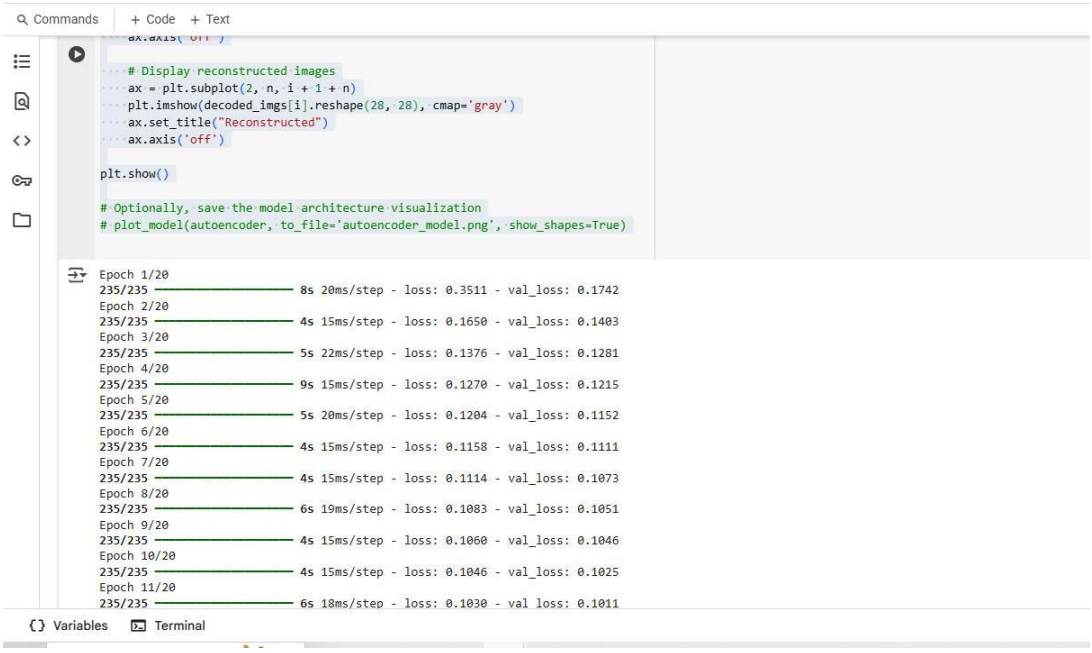
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Input, Dense
from keras.optimizers import Adam
from keras.utils import plot_model

# Step 1: Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize the images to the range [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Flatten the images to 1D vectors (28x28 -> 784)
x_train = x_train.reshape((x_train.shape[0], 784))
x_test = x_test.reshape((x_test.shape[0], 784))

# Step 2: Build the Autoencoder model
input_img = Input(shape=(784,))

# Encoder: Compress the data to a lower-dimensional space
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)    # Bottleneck layer
(compressed representation)

# Decoder: Reconstruct the data back to its original shape
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)    # Output should have
the same shape as input (784)

# Combine the encoder and decoder to form the autoencoder
autoencoder = Model(input_img, decoded)

# Step 3: Compile the model
autoencoder.compile(optimizer=Adam(), loss='binary_crossentropy')
```

```
# Step 4: Train the Autoencoder
autoencoder.fit(x_train, x_train, epochs=20, batch_size=256, shuffle=True,
validation_data=(x_test, x_test))

# Step 5: Evaluate the Autoencoder's performance on the test set
decoded_imgs = autoencoder.predict(x_test)

# Step 6: Visualize the original and reconstructed images
n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original images
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    ax.set_title("Original")
    ax.axis('off')

    # Display reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    ax.set_title("Reconstructed")
    ax.axis('off')

plt.show()

# Optionally, save the model architecture visualization
# plot_model(autoencoder, to_file='autoencoder_model.png',
show_shapes=True)
```



```
# Decoder: Reconstruct the data back to its original shape
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)  # Output should have the same shape as input (784)

# Combine the encoder and decoder to form the autoencoder
autoencoder = Model(input_img, decoded)

# Step 3: Compile the model
autoencoder.compile(optimizer=Adam(), loss='binary_crossentropy')

# Step 4: Train the Autoencoder
autoencoder.fit(x_train, x_train, epochs=20, batch_size=256, shuffle=True, validation_data=(x_test, x_test))

# Step 5: Evaluate the Autoencoder's performance on the test set
decoded_imgs = autoencoder.predict(x_test)

# Step 6: Visualize the original and reconstructed images
n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original images
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    ax.set_title("Original")
    ax.axis('off')

    # Display reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    ax.set_title("Reconstructed")
    ax.axis('off')
```



```
    # Display reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    ax.set_title("Reconstructed")
    ax.axis('off')

plt.show()

# Optionally, save the model architecture visualization
# plot_model(autoencoder, to_file='autoencoder_model.png', show_shapes=True)
```

```
Epoch 1/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - loss: 0.3511 - val_loss: 0.1742
Epoch 2/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1650 - val_loss: 0.1403
Epoch 3/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.1376 - val_loss: 0.1281
Epoch 4/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 15ms/step - loss: 0.1270 - val_loss: 0.1215
Epoch 5/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.1204 - val_loss: 0.1152
Epoch 6/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1158 - val_loss: 0.1111
Epoch 7/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1114 - val_loss: 0.1073
Epoch 8/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.1083 - val_loss: 0.1051
Epoch 9/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1060 - val_loss: 0.1046
Epoch 10/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1046 - val_loss: 0.1025
Epoch 11/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - loss: 0.1030 - val_loss: 0.1011
```

Output:


Commands
+ Code
+ Text
Connect
Epoch 13/20
235/235 6s 19ms/step - loss: 0.1006 - val_loss: 0.0988
Epoch 14/20
235/235 4s 15ms/step - loss: 0.0994 - val_loss: 0.0975
Epoch 15/20
235/235 5s 15ms/step - loss: 0.0981 - val_loss: 0.0961
Epoch 16/20
235/235 4s 19ms/step - loss: 0.0970 - val_loss: 0.0949
Epoch 17/20
235/235 4s 15ms/step - loss: 0.0960 - val_loss: 0.0945
Epoch 18/20
235/235 6s 17ms/step - loss: 0.0948 - val_loss: 0.0936
Epoch 19/20
235/235 4s 18ms/step - loss: 0.0945 - val_loss: 0.0932
Epoch 20/20
235/235 4s 15ms/step - loss: 0.0942 - val_loss: 0.0929
WARNING:tensorflow:5 out of the last 15 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7b8da1473e20> triggered tf.function retracing.
313/313 1s 2ms/step
Original
Reconstructed
Variables
Terminal

PRACTICAL 9.

Aim: Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.(google stock price).

Code:

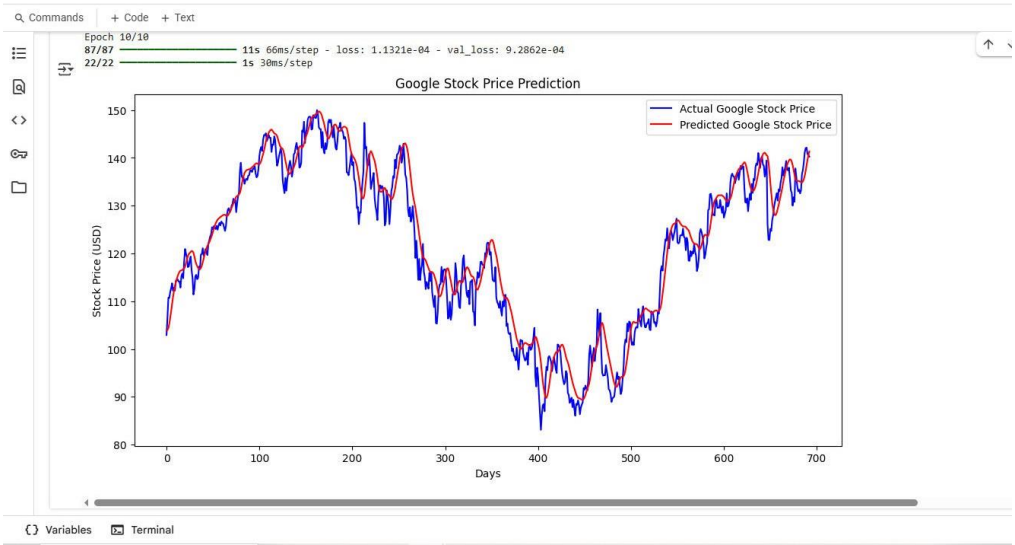
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Step 1: Load historical stock data
data = yf.download('GOOG', start='2010-01-01', end='2023-12-31')
stock_prices = data['Close'].values.reshape(-1, 1)    # Use 'Close' price

# Step 2: Normalize prices
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_prices = scaler.fit_transform(stock_prices)

# Step 3: Create sequences for training (60 days to predict next day)
X = []
y = []
sequence_length = 60

for i in range(sequence_length, len(scaled_prices)):
    X.append(scaled_prices[i-sequence_length:i])
    y.append(scaled_prices[i])

X = np.array(X)
y = np.array(y)

# Step 4: Split into training and testing sets (80% training)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Step 5: Build the LSTM model
model = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
1)),
    LSTM(units=50),
```

```
    Dense(units=1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

# Step 6: Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))

# Step 7: Predict and inverse scale
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
actual = scaler.inverse_transform(y_test.reshape(-1, 1))

# Step 8: Plot results
plt.figure(figsize=(12, 6))
plt.plot(actual, color='blue', label='Actual Google Stock Price')
plt.plot(predictions, color='red', label='Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Days')
plt.ylabel('Stock Price (USD)')
plt.legend()
plt.show()
```

```python
# Step 4: Split into training and testing sets (80% training)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Step 5: Build the LSTM model
model = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
    LSTM(units=50),
    Dense(units=1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

# Step 6: Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Step 7: Predict and inverse scale
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
actual = scaler.inverse_transform(y_test.reshape(-1, 1))

# Step 8: Plot results
plt.figure(figsize=(12, 6))
plt.plot(actual, color='blue', label='Actual Google Stock Price')
plt.plot(predictions, color='red', label='Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Days')
plt.ylabel('Stock Price (USD)')
plt.legend()
plt.show()
```

```
[**********100%**********]  1 of 1 completedEpoch 1/10

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using a
  super().__init__(**kwargs)
87/87 ━━━━━━━━━━━━━━━━━━━━ 8s 55ms/step - loss: 0.0090 - val_loss: 0.0013
Epoch 2/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 6s 66ms/step - loss: 1.5237e-04 - val_loss: 0.0012
Epoch 3/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 10s 65ms/step - loss: 1.2637e-04 - val_loss: 0.0011
Epoch 4/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 11s 69ms/step - loss: 1.2209e-04 - val_loss: 0.0012
Epoch 5/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 5s 57ms/step - loss: 1.1353e-04 - val_loss: 0.0012
Epoch 6/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - loss: 1.3336e-04 - val_loss: 0.0010
Epoch 7/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 5s 63ms/step - loss: 1.1695e-04 - val_loss: 9.6799e-04
Epoch 8/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 9s 51ms/step - loss: 1.3559e-04 - val_loss: 0.0011
Epoch 9/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 5s 61ms/step - loss: 1.1140e-04 - val_loss: 8.5251e-04
Epoch 10/10
87/87 ━━━━━━━━━━━━━━━━━━━━ 11s 66ms/step - loss: 1.1321e-04 - val_loss: 9.2862e-04
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step
```

Output:



PRACTICAL 10.

Aim: Applying Generative Adversarial Networks for image generation and unsupervised tasks.

Code:

STEP 1: Import Libraries

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
import os
```

STEP 2: Load CIFAR-10 Dataset

```
(x_train, _), (_, _) = tf.keras.datasets.cifar10.load_data()
x_train = x_train.astype('float32')
x_train = (x_train - 127.5) / 127.5  # Normalize to [-1, 1]
BUFFER_SIZE = 50000
BATCH_SIZE = 128
train_dataset = tf.data.Dataset.from_tensor_slices(x_train).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

STEP 3: Define the Generator

```
def make_generator_model():
    model = tf.keras.Sequential([
        layers.Dense(8*8*256, use_bias=False, input_shape=(100,)),
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Reshape((8, 8, 256)),
        layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False),
```

```
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same', use_bias=False,
activation='tanh')
    ])
    return model
generator = make_generator_model()
STEP 4: Define the Discriminator
def make_discriminator_model():
    model = tf.keras.Sequential([
        layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[32, 32, 3]),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

        layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

        layers.Flatten(),
        layers.Dense(1)
    ])
    return model

discriminator = make_discriminator_model()
STEP 5: Loss Functions & Optimizers
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
def discriminator_loss(real_output, fake_output):
```

```
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    return real_loss + fake_loss
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
STEP 6: Define Training Loop
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, noise_dim])
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
    generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))
def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input, training=False)
    fig = plt.figure(figsize=(4, 4))
    for i in range(predictions.shape[0]):
```

```
        plt.subplot(4, 4, i + 1)
        img = (predictions[i] + 1.0) / 2.0  # Rescale from [-1,1] to [0,1]
        plt.imshow(img)
        plt.axis('off')
    plt.suptitle(f'Epoch {epoch}', fontsize=16)
    plt.show()
def train(dataset, epochs):
    for epoch in range(1, epochs + 1):
        for image_batch in dataset:
            train_step(image_batch)
        if epoch % 5 == 0 or epoch == 1:
            print(f' Epoch {epoch} completed!')
            generate_and_save_images(generator, epoch, seed)
STEP 7: Start Training
train(train_dataset, EPOCHS)
```

NURTURING POTENTIAL

SAKET GYANPEETH'S

SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Permanently Affiliated to University of Mumbai)

NAAC Accredited

Saket Vidyanagri Marg, Chinchpada Road, Katemanivali,
Kalyan (East) -421306(Mah)

Department of Information Technology

This is to certify that

Mr. SACHIDANAND SANTOSH DUBEY

Seat No. 1314296

of

M.Sc. Information Technology

Part II NEP 2020 Semester IV

has satisfactory carried out the required practical in the subject of **CYBER FORENSICS**

For the Academic year 2024 – 2025

Practical In-Charge **Head of the Department**

External Examiner

College Seal

INDEX

Sr. No.		Practical	Signature
1.	a.	**Computer Forensics Investigation Process** Recovering Data using the EaseUS Data Recovery Wizard	
	b.	Performing Hash, Checksum, or HMAC Calculations using the HashCalc.	
	c.	Creating a Disk Image File of a Hard Disk Partition using the R-drive Image Tool.	
2.	a.	**Understanding Hard Disks and File Systems** Analyzing File System Types Using the Sleuth Kit (TSK)	
	b.	Analyzing Raw image using Autopsy.	
	c.	Analyze file system of Linux image file.	
	d.	Analyze file system of Windows image file.	
3.	a.	**Data Acquisition and Duplication** Creating a dd image file	
	b.	Investigating NTFS Drive Using DiskExplorer for NTFS	
	c.	Viewing Content of Forensic Image Using Access Data FTK Imager Tool	
4.	a.	**Defeating Anti-forensics Techniques** Cracking Application Password	
	b.	Detecting Steganography	
	c.	Perform a practical of identifying the packer used to pack a file by using ExeInfo PE and then unpacking the file using UPX	
5.	a.	**Performing OS Forensics** Perform a Practical collect volatile information from a host computer running on a Windows OS by using tools PsTools, LogonSessions, and NetworkOpenedFiles	
	b.	Perform a Practical for Discovering and Extracting Hidden Forensic Material on Computers Using OSForensics	
	c.	Performing a Computer Forensic Investigation Using the Helix Tool	
	d.	Examine Windows event logs using Event Log Explorer	

6.	a.	**Network Forensics** Investigating Network Traffic Using Wireshark	
	b.	Investigating Network Attacks using Kiwi Log Viewer	
7.		**Investigating Web Attacks** Analyzing Domain and IP Address Queries Using SmartWhois Tool	
8.		**Database Forensics** Analyzing SQLite Databases using DB Browser for SQLite	
9.	a.	**Malware Forensics** Perform Static Analysis of the Suspicious File	
	b.	Performing dynamic analysis of a malicious file to find the processes It starts, network operations, file changes and other activities.	
10.	a.	**Investigating Email Crimes** Recovering Deleted Emails Using the Recover My Email utility.	
	b.	Tracing an Email Using the eMailTrackerPro Tool.	
11.		**Mobile Forensics** Analyzing the Forensic Image and Carving the Deleted Files Using Autopsy.	

COMPUTER FORENSICS INVESTIGATION PROCESS

PRACTICAL 1 A.

Aim: Recovering Data using the EaseUS Data Recovery Wizard

Code:

Computer Forensics Investigation Process & Data Recovery Using EaseUS Data Recovery Wizard

1. Overview of Computer Forensics Investigation Process

Computer forensics involves identifying, preserving, analyzing, and presenting digital evidence in a legally acceptable manner. Below is a structured process:

Step-by-Step Process:

1.1 Identification

- Determine the scope and nature of the incident.
- Identify potential sources of evidence (hard drives, cloud storage, mobile devices, etc.).

1.2 Preservation

- Isolate and protect data to prevent tampering.
- Create **bit-by-bit forensic images** using tools like FTK Imager or dd.
- Maintain a **chain of custody**.

1.3 Collection

- Extract data from digital sources securely.
- Ensure data integrity using hash functions (MD5, SHA-1).

1.4 Examination

- Search and filter data to locate relevant evidence.
- Identify deleted files, hidden partitions, and encrypted data.

1.5 Analysis

- Reconstruct events, timelines, or data activities.
- Link recovered data to specific users or activities.

1.6 Documentation & Reporting

- Document all procedures and findings.
- Prepare a formal report for stakeholders or legal authorities.
- Maintain proper evidence handling logs.

1.7 Presentation

- Present findings in court if required.
- Be prepared to testify as an expert witness.

2. Recovering Data Using EaseUS Data Recovery Wizard

EaseUS Data Recovery Wizard is a powerful tool for recovering lost, deleted, or formatted data.

Step-by-Step Recovery Process:

2.1 Install the Software

- Download and install **EaseUS Data Recovery Wizard** from the official website.
- Avoid installing it on the drive from which you want to recover data to prevent overwriting.

2.2 Launch the Program

- Open the application. It will display all available drives and partitions.

2.3 Select the Drive/Location

- Choose the location where you lost the data (e.g., hard disk partition, USB drive, or Recycle Bin).
- Click **Scan**.

2.4 Scanning Process

- **Quick Scan**: Finds recently deleted files.
- **Deep Scan**: Thorough scan to find files from formatted, damaged, or raw partitions. This takes more time.

2.5 Preview and Recover Files

- Once the scan is complete, browse through the results.
- Use filters or search by file type (e.g., documents, photos, videos).
- Preview files to confirm they are recoverable.

2.6 Recover Data

- Select the files you want to recover.

- Click **Recover** and choose a safe recovery location (not the original drive to avoid data overwriting).

3. Important Considerations in Forensics

- **Data Integrity:** Always create an image before any analysis or recovery.
- **Logs & Chain of Custody:** Document every step.
- **Legal Compliance:** Ensure tools and processes are court-admissible.

4. Combining EaseUS with Forensics

Although EaseUS is not a forensics-specific tool, it can be useful in the **examination** and **analysis** phases, especially for:

- Recovering deleted documents.
- Finding data from formatted drives.
- Recovering from partitions that no longer show in the OS.

For forensic-grade data recovery, combine EaseUS with tools like:

- EnCase
- Autopsy (Sleuth Kit)
- FTK (Forensic Toolkit)

PRACTICAL 1 B.

Aim: Performing Hash, Checksum, or HMAC Calculations using the HashCalc.

Code:

Here's a structured explanation and guide you can use in a report or as a reference for **Performing Hash, Checksum, or HMAC Calculations Using HashCalc**:

HashCalc is a free and easy-to-use tool for calculating hash values, checksums, and HMACs (Hash-based Message Authentication Codes). It's commonly used in digital forensics and data integrity verification.

1. What Is HashCalc?

- **Developer**: SlavaSoft
- **Purpose**: To compute checksums, hashes (MD5, SHA-1, SHA-256, etc.), and HMACs from files or text inputs.
- **Use Case in Forensics**:
  - Verify file integrity
  - Validate forensic disk images
  - Detect unauthorized changes

2. Supported Algorithms

HashCalc supports multiple algorithms, including:

- **Checksum**: CRC32, ADLER32
- **Hash Functions**: MD2, MD4, MD5, SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD160, etc.
- **HMAC**: Keyed hash using any of the supported algorithms

3. How to Use HashCalc

Step 1: Download and Install HashCalc

- Download from SlavaSoft or trusted software sites.
- Install the application on your forensic workstation.

Step 2: Launch the Application

- Open HashCalc. You'll see a GUI with the following fields:
  - **Data Format** (File, Text, Hex)
  - **Data** (input box or file selector)
  - **Key** (for HMAC only)
  - **Hash Options** (list of checkboxes for algorithms)

Step 3: Configure Your Calculation

- **Choose Input Type**: Select "File" if hashing a document, image, or executable. For short inputs, choose "Text".
- **Enter or Browse for Data**: Use the "…" button to select the file or type text directly.
- **Select Algorithms**: Check the boxes for the hash/checksum algorithms you want to compute (e.g., MD5, SHA-256).
- *(Optional)* **Enter HMAC Key**: If using HMAC, input your secret key.

Step 4: Click 'Calculate'

- Press the **Calculate** button.
- Results will appear in the bottom section for all selected algorithms.

Step 5: Save or Copy the Output

- Right-click to copy individual hashes.
- Optionally save or log the results for forensic documentation.

PRACTICAL 1 C.

Aim: Creating a Disk Image File of a Hard Disk Partition using the R-drive Image Tool.

Code:

1. Overview of R-Drive Image

R-Drive Image is a powerful disk imaging and backup tool. It enables users to create exact byte-by-byte images of entire hard drives or individual partitions. In computer forensics, disk imaging is critical for preserving digital evidence without altering the original media.

✓ Key Features:

- Creates exact sector-level images
- Supports compressed and encrypted images
- Allows mounting and restoring of images
- Can image live systems (hot imaging)

2. Importance in Digital Forensics

Creating a disk image ensures:

- **Data integrity** — original media remains untouched
- **Repeatability** — investigators can work on copies
- **Evidence admissibility** — follows legal standards

3. Steps to Create a Disk Image Using R-Drive Image

Step 1: Install and Launch the Tool

- Download and install **R-Drive Image** from the official website.
- Run the application with administrative privileges.

Step 2: Start the Disk Image Creation Wizard

- On the main screen, choose **"Create Image"**.
- The wizard will open to guide you through the imaging process.

Step 3: Select the Source Disk or Partition

- Select the disk or specific partition you want to image.
- Click **Next**.

Step 4: Choose Image Destination

- Select a destination folder to store the image file.
- Choose a filename and format (default is `.rdr`).
- It's best practice to store the image on an external drive or forensic server.

Step 5: Configure Image Options

- **Image Type**: Choose whether to image:
  - The entire disk
  - Specific partitions
- **Compression**: Optional — reduces file size.
- **Split Image** (optional): Useful if storing on FAT32 drives or DVDs.
- **Password Protection** (optional): Add encryption for security.

Step 6: Start Imaging

- Review your settings.
- Click **Start** to begin the imaging process.
- The progress bar will display the time and status.

Step 7: Verify the Image (Recommended)

- After creation, use R-Drive Image to **verify the image**.
- This ensures no corruption occurred during the imaging process.

Step 8: Document the Imaging Process

Record the following in your forensic report:

- Device details (model, serial number)
- Hash values (e.g., SHA-256) of the original and image file
- Time and date
- Investigator details
- Tools and settings used

UNDERSTANDING HARD DISKS AND FILE SYSTEMS

PRACTICAL 2 A.

Aim: Analyzing File System Types Using the Sleuth Kit (TSK)

Code:

1. Overview of The Sleuth Kit (TSK)

The Sleuth Kit (TSK) is a collection of open-source command-line tools used for digital forensic analysis of disk images and file systems. It supports analysis of common file systems like:

- **FAT (FAT12, FAT16, FAT32)**
- **NTFS**
- **ext2/ext3/ext4 (Linux)**
- **HFS+ (macOS)**
- **exFAT, ISO9660**, and more.

2. Purpose of File System Analysis in Forensics

File system analysis helps in:

- Recovering deleted files
- Understanding file metadata (timestamps, access logs)
- Identifying hidden or suspicious files
- Reconstructing user activity timelines

3. Common TSK Tools for File System Analysis

Tool	Purpose
`fsstat`	Displays detailed information about the file system
`fls`	Lists files and directories in a file system
`istat`	Displays details of a file's metadata (inode)
`blkls`	Extracts unallocated space (for data carving)

Tool	Purpose
`ils`	Lists all inodes
`icat`	Extracts file content based on inode
`mmls`	Lists partition layout of a disk image

4. Steps to Analyze a File System Using TSK

Step 1: Mount or Examine the Disk Image

Assume the image is `image.dd`. First, check the partition layout:

```
mmls image.dd
```

Note the start sector of the partition you want to analyze.

Step 2: Get File System Metadata

Use `fsstat` with the appropriate offset:

```
fsstat -o 2048 image.dd
```

Replace `2048` with the correct sector offset. This provides:

- File system type (e.g., NTFS, ext4)
- Volume serial number
- Cluster size, block size
- File creation and modification timestamps
- Allocation methods

Step 3: List Files and Directories

Use `fls` to list the contents of the file system:

```
fls -r -o 2048 image.dd > filelist.txt
```

- `-r` lists recursively
- Output is saved to `filelist.txt` for review

Step 4: View Specific File Metadata

Find a specific inode number from `fls` output and use `istat`:

```
istat -o 2048 image.dd 128
```

This shows:

- Timestamps (Created, Modified, Accessed)
- File size
- Data block locations

Step 5: Recover Deleted Files

If the file is marked as deleted in `fls` output (e.g., `*`), you can recover it:

```
icat -o 2048 image.dd 128 > recovered.docx
```

Step 6: Analyze Unallocated Space

You can extract unallocated space for carving or further analysis:

```
blkls -o 2048 image.dd > unallocated.raw
```

Then analyze it with carving tools like `foremost` or `photorec`.

PRACTICAL 2 B.

Aim: Analyzing Raw image using Autopsy.

Code:

1. Overview of Autopsy

Autopsy is a graphical interface for **The Sleuth Kit (TSK)**, widely used in digital forensics. It allows investigators to analyze disk images, recover deleted files, and view user activity in a timeline format — all within a user-friendly GUI.

✅ Key Features:

- Timeline analysis
- File system browsing
- Keyword search
- Email and web artifacts recovery
- Deleted file recovery
- Registry and user activity analysis (for Windows systems)

2. What Is a Raw Image?

A **raw image** (`.dd`, `.img`, or `.raw`) is an exact bit-by-bit copy of a storage device or partition, including unallocated and slack space.

3. Steps to Analyze a Raw Image Using Autopsy

Step 1: Install Autopsy

- Download from https://www.autopsy.com
- Available for Windows and Linux
- Java runtime included in the installer

Step 2: Launch Autopsy and Create a New Case

1. Open Autopsy.
2. Click **"Create New Case"**.
3. Enter:
    - Case Name
    - Case Number (optional)
    - Examiner Name

4. Choose a case folder location.
5. Click **Finish**.

Step 3: Add a Data Source

1. Click **"Add Data Source"**.
2. Select **"Disk Image or VM File"**.
3. Browse and select your raw image file (e.g., `evidence.dd`).
4. Specify the image type (e.g., raw/dd).
5. Click **Next**, then configure ingest modules.

Step 4: Configure Ingest Modules

Autopsy will offer to run modules for:

- File Type Identification
- Recent Activity
- Hash Lookup
- Keyword Search
- Email Parser
- Web History, etc.

Step 5: Begin Analysis

After indexing is complete, use the left-side navigation to explore:

Section	Purpose
Data Sources	View files and folders in the image
Views	Browse by type (images, docs, videos, etc.)
Keyword Hits	View keyword search results
Recent Activity	Review user actions (browser, documents, USBs, etc.)
Deleted Files	Recovered files no longer visible in the OS
File Metadata	Access timestamps, hashes, MIME type, etc.

Step 6: Use Timeline (Optional)

1. Go to the **Timeline** tab.
2. Filter by:
    - File creation/modification/access

    - Time range
    - Specific user activities
3. This helps reconstruct events like file access or browser history.

Step 7: Export and Document Findings

- Export files, reports, screenshots, or evidence summaries.
- Use built-in report generation (PDF/HTML/Excel).
- Document:
    - File paths
    - Hashes
    - Metadata (timestamps, size)
    - Module results

4. Best Practices for Autopsy Usage

Best Practice	Why It Matters
Use verified hash values of the image	Ensure integrity of analysis
Work on a forensic image, not the original	Maintain evidence admissibility
Document every step taken	Ensures chain of custody and transparency
Run only relevant modules	Saves time and reduces noise

PRACTICAL 2 C.

Aim: Analyze file system of Linux image file.

Code:

1. Identify the File System Type

Tool: `file`, `fdisk`, `mmls`, or `parted`

`file linux_image.dd`
This gives you the file type and potentially the file system type (e.g., ext4).
To list partitions inside the image:
`mmls linux_image.dd`
Note the **start sector** of the Linux partition.

2. Mount the Linux Partition (Optional Preview)

If you need to mount it manually (not recommended for write-safe forensics), use:

```
sudo mkdir /mnt/linux_image
sudo mount -o ro,loop,offset=$((512*START_SECTOR)) linux_image.dd
/mnt/linux_image
```

Replace `START_SECTOR` with the number found using `mmls`.

3. Analyze with Sleuth Kit (TSK) Tools

Assume the partition starts at sector `2048`.

a. File System Metadata

```
fsstat -o 2048 linux_image.dd
```

Gives details about:

- File system type
- Inode structure
- Allocation info
- Volume label, timestamps

b. List All Files (Including Deleted)

```
fls -r -o 2048 linux_image.dd > linux_fls.txt
```

Flags:

- `-r`: recursive
- `-o`: sector offset

This gives a timeline of file creation and deletion. Deleted files are marked with `*`.

c. View Metadata of a Specific File

```
istat -o 2048 linux_image.dd 128
```
Where `128` is the inode number (found using `fls`).

d. Recover Files or Data

```
icat -o 2048 linux_image.dd 128 > recovered_file
```
Recovers the file content from a specific inode.

e. Extract Unallocated Space (for Data Carving)

```
blkls -o 2048 linux_image.dd > unallocated.raw
```
Feed `unallocated.raw` to tools like `photorec` or `foremost` to carve files.

4. Analyze User Activity (Linux Artifacts)

Once you've listed or mounted the image, check common locations:

Location	Purpose
`/home/username/`	User documents and browser data
`/var/log/`	System logs (auth, syslog, dmesg)
`/etc/passwd` & `/etc/shadow`	User accounts and password hashes
`.bash_history`, `.ssh/`, `.config/`	User command history and sessions

5. Document and Report

Record:

- File system type
- Sector offset and mount details
- Hashes of files or extracted data
- Inodes, paths, and timestamps
- Any findings (e.g., suspicious scripts, malware, logs)

PRACTICAL 2 D.

Aim: Analyze file system of Windows image file.

Code:

1. Tools You Can Use

- **The Sleuth Kit (TSK)** – CLI-based analysis
- **Autopsy** – GUI interface for TSK
- **FTK Imager** – For viewing and exporting files
- **X-Ways Forensics / EnCase** – Commercial tools
- **Log2timeline / Plaso** – Timeline analysis
- **Volatility** – For memory and hibernation file analysis

2. Determine Partition Layout

Use `mmls` (from Sleuth Kit):

```
mmls windows_image.dd
```

This shows the partitions in the image. Look for the NTFS partition and note the **Start Sector**.

3. Analyze File System Metadata with TSK

a. View File System Info

```
fsstat -o START_SECTOR windows_image.dd
```

This reveals:

- File system type (e.g., NTFS)
- Volume serial number
- MFT (Master File Table) details
- Allocation size

b. List Files and Directories

```
fls -r -o START_SECTOR windows_image.dd > fls_output.txt
```

- `-r`: recursive
- Deleted files are marked with `*`

This gives a complete directory tree with inode numbers and file metadata.

c. Examine a File's Metadata

```
istat -o START_SECTOR windows_image.dd INODE_NUMBER
```

Shows:

- MAC times (Modified, Accessed, Changed)
- File size, flags, attributes
- Logical/physical data blocks

d. Recover Files by Inode

```
icat -o START_SECTOR windows_image.dd INODE_NUMBER > recovered_file
```

Use this to recover deleted or intact files from the image.

4. Key Forensic Artifacts in Windows

Once mounted or extracted, check these directories:

Path	Artifact
`C:\Users\`	User profiles, documents, app data
`C:\Users\%username%\AppData\`	Browser data, recent files, program configs
`C:\Windows\System32\config`	Registry hives (SYSTEM, SAM, SOFTWARE)
`C:\Windows\Prefetch\`	App execution history
`C:\Windows\System32\winevt\Logs\`	Event logs (security, application, system)
`pagefile.sys` / `hiberfil.sys`	Memory artifacts

Use tools like:

- **Registry Explorer** to read registry hives
- **Event Viewer** or **EVTXtract** to analyze event logs

5. Timeline Creation (Optional Advanced)

To generate a timeline from file metadata:

Use `log2timeline` (Plaso)

```
log2timeline.py timeline.plaso windows_image.dd
```

Then view it using:

```
psort.py -o l2tcsv -w timeline.csv timeline.plaso
```

This provides a full activity log (file creation, access, login times, etc.).

6. Documentation and Reporting

For forensic reports, document:

- File system type and partition layout
- Any deleted or suspicious files
- User activity (logins, file access, browser history)
- MAC times, hash values, and recovery steps
- Tools and commands used

DATA ACQUISITION AND DUPLICATION

PRACTICAL 3 A.

Aim: Creating a dd image file
Code:

Why Use `dd` in Forensics?

- Creates an exact, low-level copy (including unallocated space and deleted files)
- Supports full disk or partition imaging
- Preserves metadata and timestamps
- Ideal for forensic analysis and evidence preservation

Before You Begin

Step	Recommendation
Use a **write blocker**	Prevents accidental writing to the original media
Work as **root/admin**	Needed for direct device access
Identify the correct drive	Mistaking the drive can overwrite valuable data

Use `lsblk`, `fdisk -l`, or `df -h` to identify device names.

Syntax of `dd` Command

```
dd if=/dev/sdX of=/path/to/output.img bs=4M status=progress
```

Parameter	Description
`if=`	Input file/device (e.g., `/dev/sdb`)
`of=`	Output file (e.g., `evidence.dd`)
`bs=4M`	Block size (4MB is efficient)
`status=progress`	Shows progress in real-time

Example: Create an Image of a USB Drive

1. **Identify the drive:**

```
sudo fdisk -l
```

Assume it is `/dev/sdb`.

2. **Create the image:**

```
sudo dd if=/dev/sdb of=/mnt/forensics/usb_image.dd bs=4M status=progress
```

3. **Verify the image using a hash:**

```
md5sum /dev/sdb
md5sum /mnt/forensics/usb_image.dd
```

Hashes should match to confirm integrity.

Optional Flags for Forensics

- `conv=noerror,sync`: Continues even if errors are encountered

```
dd if=/dev/sdb of=usb_image.dd bs=4M conv=noerror,sync status=progress
```

- `count=`: To image only a portion of the disk (e.g., first 1 GB)

```
dd if=/dev/sdb of=sample_image.dd bs=1M count=1024
```

Documentation Checklist

Item	Example
Device Name	`/dev/sdb`
Image File Name	`usb_image.dd`
Date and Time of Acquisition	`2025-05-30 10:30 AM UTC`
Hash Values (MD5/SHA256)	`e99a18c428cb38d5f260853678922e03`
Investigator Name	Your Name
Tool and Command Used	`dd if=/dev/sdb of=usb_image.dd bs=4M ...`

PRACTICAL 3 B.

Aim: Investigating NTFS Drive Using Disk Explorer for NTFS

Code:

1. Overview of Disk Explorer for NTFS

Disk Explorer for NTFS is a forensic utility that allows users to explore the internal structures of NTFS-formatted drives at a low level. It enables analysts to view:

- Boot sector
- Master File Table (MFT)
- File records
- Deleted files
- Slack and unallocated space

It is especially useful when you need to:

- Recover deleted files manually
- Investigate hidden data
- Analyze metadata (timestamps, permissions, flags)

2. Key NTFS Structures You Can Analyze

Structure	Purpose
Boot Sector (Volume Boot Record)	Contains disk layout and partition info
Master File Table (MFT)	Core of NTFS – stores metadata about every file
File Records	Individual entries for files (name, size, MAC times)
Data Runs	Logical to physical mapping of file data
Slack Space	May contain remnants of previously deleted files
Unallocated Space	Unused but potentially recoverable data area

3. Step-by-Step: Using Disk Explorer for NTFS

✓ Step 1: Launch the Program

- Open **Disk Explorer for NTFS** (often part of runtime disk utilities or forensic toolkits).
- Ensure you're running it with **administrative privileges** to access physical drives.

✅ Step 2: Select the NTFS Drive

- Choose the physical or logical NTFS volume you want to investigate.
- Disk Explorer shows a hierarchical view of the NTFS structure.

✅ Step 3: Explore the File System

View MFT Entries

- Navigate to the **MFT** (Master File Table).
- You can see individual records for files, folders, and system metadata.

Inspect File Attributes

- For any file, view attributes such as:
  - **$STANDARD_INFORMATION** (MAC times, permissions)
  - **$FILE_NAME** (timestamps from directory entry)
  - **$DATA** (file content or pointer to data runs)
  - **$BITMAP**, **$INDEX_ROOT**, **$INDEX_ALLOCATION** (for directories)

Analyze Deleted Files

- Disk Explorer often marks deleted entries in red or with a strikethrough.
- Check if the clusters have been overwritten; if not, the file can be recovered.

✅ Step 4: Recover Data

- Right-click on a deleted file entry and choose **Recover**.
- Export the file to a different disk (not to the source drive).

✅ Step 5: View Raw Data

- Open any file entry in **hex view**.
- Useful for:
  - Viewing embedded metadata
  - Identifying hidden file content
  - Carving file headers/footers manually

4. Forensic Considerations

Aspect	Best Practice
Read-only analysis	Work on an image or write-blocked drive
Hash verification	Hash original disk/image before and after analysis
Chain of custody	Log actions, timestamps, and tools used
Metadata preservation	Always document MAC times and attribute changes

PRACTICAL 3 C.

Aim: Viewing Content of Forensic Image Using Access Data FTK Imager Tool

Code:

1. What is FTK Imager?

FTK Imager is a widely used forensic tool for **creating, analyzing, and verifying disk images**. It allows forensic investigators to examine disk images, extract evidence, and verify the integrity of the data. It supports a variety of image formats, including `.dd`, `.E01`, `.img`, and `.ISO`.

With FTK Imager, you can:

- View the contents of forensic images.
- Recover deleted files.
- Extract files and folders.
- Verify file integrity using hashes (MD5, SHA-1, SHA-256).
- Perform a detailed forensic examination of file systems (e.g., NTFS, FAT32).

2. Installing FTK Imager

If you don't have **FTK Imager** installed:

1. Download it from AccessData's official website.
2. Install the tool on your forensic workstation, ensuring you run it with **administrative privileges**.

3. Step-by-Step Guide to Viewing the Content of a Forensic Image

✓ Step 1: Launch FTK Imager

1. Open **FTK Imager** on your computer.
2. Click on **File > Add Evidence Item** to begin the process of loading a forensic image.

✓ Step 2: Load the Forensic Image

1. **Select the evidence type**:
   - If you are working with a disk image, choose **"Image File"**.
   - FTK Imager supports multiple image formats, including `.dd`, `.E01`, `.img`, etc.
2. **Browse for the image file** you want to analyze (e.g., `evidence_image.E01`).
3. **Open the image** by selecting it and clicking **OK**.

✅ Step 3: Analyze the Disk Image

Once the forensic image is loaded, FTK Imager will display the following details:

1. **File System View**: FTK Imager displays the image's file system structure (directories, subdirectories, files, etc.).
2. **Directory Tree View**: On the left side of the window, you will see the directory structure of the image, similar to a file explorer. It shows folders, files, and other file system metadata.
3. **File Metadata**: Clicking on any file will display its metadata in the bottom section:
    - **File name**
    - **File size**
    - **Creation, modified, and last access dates (timestamps)**
    - **File type (extension)**
    - **File attributes (hidden, read-only, etc.)**

✅ Step 4: Preview the Files

1. **Preview Text Files**: For text-based files (e.g., `.txt`, `.html`, `.csv`), FTK Imager will show the file contents directly within the program.
2. **Preview Binary Files**: For binary files (e.g., images, executables), FTK Imager may not display the file's content directly, but you can still extract them.
3. **Hex View**: FTK Imager allows you to open files in **hexadecimal view**, which is particularly useful when analyzing:
    - Non-text files
    - Partially corrupted files
    - Embedded data within the file's header/footer

✅ Step 5: Search and Filter Data

1. **Search Files**: Use the **Search** function to find specific files or keywords. You can search for:
    - **Keywords**: Specific text or phrases.
    - **File Types**: Search by file extensions (e.g., `.jpg`, `.exe`).
    - **File Names**: Search by the name of the file or folder.
2. **Filtering Results**: You can filter search results by:
    - **Date ranges** (creation, modification, access dates)
    - **File size**
    - **File attributes** (hidden, system, etc.)

✅ Step 6: Recover Deleted Files

1. **Deleted Files**: FTK Imager marks deleted files (if recoverable) with a red icon or strikethrough.
2. **Recover Deleted Files**:

- Right-click on the deleted file.
- Select **"Export"** to recover the file.
- Save the file to a different location to avoid modifying the original image.

✅ Step 7: Export Evidence

1. **Export Files/Folders**: You can export individual files or entire folders to a separate location.
   - Right-click on the file or folder.
   - Choose **"Export Files"** or **"Export Directory"**.
   - Specify the export location.
2. **Hash Exported Files**: You can compute hashes (MD5, SHA-1, SHA-256) for any exported file to verify its integrity.

✅ Step 8: Verifying Integrity (Hashing)

1. **Hash the Image**: To ensure the integrity of the forensic image, FTK Imager allows you to generate hash values (MD5, SHA-1, SHA-256).
2. **Steps for Hashing**:
   - Go to **File > Hash File**.
   - Select the image you are analyzing.
   - Choose the hashing algorithm (MD5, SHA-1, or SHA-256).
   - FTK Imager will compute and display the hash value for the image.

4. Forensic Considerations

Consideration	Best Practice
Write Protection	Always use a **write blocker** to prevent modifying the source image.
Chain of Custody	Record each action taken with the evidence (date, time, actions).
Image Integrity	Hash the image before and after analysis to ensure no data tampering.
Evidence Export	Ensure that recovered or extracted files are saved to a separate drive, not the original image.

DEFEATING ANTI-FORENSICS TECHNIQUES

PRACTICAL 4 A.

Aim: Cracking Application Password

Code:

Part 1: Defeating Anti-Forensics Techniques

Anti-forensics are methods attackers use to hide, alter, or destroy evidence. Detecting and overcoming these techniques is vital in forensic investigations.

Common Anti-Forensics Techniques and How to Defeat Them

Technique	Description	Countermeasure/Method
Data Wiping/Overwriting	Using tools to securely erase data.	Use forensic tools to recover overwritten data (e.g., data carving with **Autopsy**, **Scalpel**).
File Timestamp Manipulation	Changing file creation/modification/access times.	Compare timestamps from multiple sources (filesystem metadata, logs). Use hash-based timeline analysis.
Encryption	Encrypting files or disk volumes.	Use password cracking (if legal) or exploit vulnerabilities; analyze encrypted containers with known keys or attempts to brute force passwords.
Data Hiding (Steganography)	Hiding data inside images, audio, or unused spaces.	Use steganalysis tools (e.g., **Stegdetect**, **OpenStego**). Analyze slack space and unallocated clusters.
Anti-Forensics Tools	Tools like Timestomp, CCleaner, or BleachBit to clean traces.	Document tool signatures; analyze artifacts left behind. Use write blockers to preserve evidence.
Log Cleaning/Deletion	Deleting or tampering with system/application logs.	Recover deleted logs using file carving or shadow copies; analyze network logs and other correlated evidence sources.

Part 2: Cracking Application Passwords

Goal: Extract or recover passwords to access protected data in a forensic investigation.

Step-by-Step Guide

1. Identify the Application and Password Storage

- Determine the application type (e.g., web browser, email client, proprietary app).
- Identify where passwords are stored:
  - Plaintext files or config files
  - Encrypted files or databases (e.g., SQLite)
  - Registry entries (Windows)
  - Memory dumps

2. Extract Password Hashes or Encrypted Passwords

- Use specialized tools or manual extraction:
  - **Browser passwords**: Use tools like **Browser Password Dump** or **NirSoft WebBrowserPassView**.
  - **Windows hashes**: Use **Mimikatz** or **pwdump**.
  - **Database passwords**: Export database and extract hashes.

3. Use Password Cracking Tools

- **John the Ripper** or **Hashcat** for cracking hashes.
- Select appropriate attack mode:
  - **Dictionary attack** with wordlists like **rockyou.txt**
  - **Brute-force attack** for shorter passwords
  - **Hybrid attack** (dictionary + mutations)

Example (Hashcat):

```
hashcat -m [hash_type] -a 0 hashes.txt rockyou.txt
```

4. Use Password Recovery or Reset (If Applicable)

- For some applications, reset options or recovery mechanisms can be used ethically.

5. Analyze Memory Dumps for Plaintext Passwords

- Tools: **Volatility Framework** for memory forensics.
- Extract plaintext credentials from memory.

Tools Summary for Practical

Tool Name	Purpose
Autopsy	Forensic analysis, carving deleted files
Stegdetect/OpenStego	Detect steganography
Mimikatz	Extract Windows credentials from memory
John the Ripper	Password hash cracking
Hashcat	GPU-accelerated password cracking
Volatility	Memory forensic analysis
NirSoft Tools	Browser password extraction

Sample Practical Scenario

Scenario: You have an encrypted zip file with a password. You suspect the attacker used anti-forensics to clean logs and timestamps.

1. Create a forensic disk image.
2. Use **Autopsy** to analyze the image, carve for deleted files or logs.
3. Extract the zip file password hash or encrypted metadata.
4. Use **Hashcat** with a dictionary attack to recover the password.
5. Document timestamps and logs; check for evidence of timestomping.
6. Report findings with screenshots, commands used, and hashes verified.

PRACTICAL 4 B.

Aim: Detecting Steganography

Code:

Detecting steganography in a forensic investigation can be a complex task, as it involves uncovering hidden data embedded within seemingly innocent files (e.g., images, audio, video). The goal of steganography detection is to identify these hidden files, extract the concealed data, and analyze it for evidence.

1. What is Steganography?

Steganography is the practice of concealing information within other, non-suspicious files such as images, audio, video, or text. The goal is to hide the existence of the message so that it remains undetected.

Types of Steganography:

- **Image Steganography**: Hiding data in the least significant bits (LSB) of pixel values.
- **Audio Steganography**: Concealing information within sound files (e.g., WAV, MP3).
- **Video Steganography**: Embedding hidden data within frames of a video.
- **Text Steganography**: Hiding information in text files through formatting or invisible characters.

2. Common Tools for Detecting Steganography

Here are some tools that can help in detecting steganography:

Tool Name	Description	Use Case
Stegdetect	Detects steganography in JPEG images using a statistical approach.	Detect hidden data in image files.
StegExpose	A Java-based tool for detecting LSB steganography in image files.	Analyze images for LSB-based steganography.
X1 Social Discovery	Used for discovering and extracting steganographic data in social media files and images.	Social media file analysis.
OpenStego	Open-source tool for both embedding and detecting data in images.	Detect and extract hidden data.

Tool Name	Description	Use Case
Binwalk	Scans files for embedded data (e.g., hidden files or images) inside larger files.	Detect hidden files in images or firmware.
Foremost	File carving tool that can extract hidden files from unallocated space.	Recover hidden or deleted data.
Sleuth Kit/Autopsy	Digital forensics toolkit for carving and analyzing hidden data in file systems.	Detect hidden data in disk images.

Detecting Steganography in Images

Step 1: Image Analysis (Metadata and Size Inspection)

1. **Check for unusual file sizes**: Steganography often results in file size changes that don't correlate with the visible content (e.g., an image may appear normal but is significantly larger than usual).
    - **Tool**: Use **ExifTool** to inspect metadata and size.
    - `exiftool image.jpg`
    - Look for suspicious changes in the **file size**, **resolution**, and **timestamp**.
2. **Check for embedded metadata**:
    - Sometimes, steganographic data is hidden in metadata fields like comments or EXIF tags.
    - **Tool**: **ExifTool** or **StegExpose**.

Step 2: Statistical Analysis (LSB Detection)

1. **Detecting Least Significant Bit (LSB) Manipulation**: The most common method for embedding data in images is modifying the least significant bit of each pixel's RGB values.
    - **Tool**: **Stegdetect** can be used to detect statistical anomalies indicative of LSB steganography.
    - `stegdetect image.jpg`
2. **Visual Inspection for Artifacts**: Use a **histogram** or **contrast adjustment** to visually detect differences in pixel distribution that could indicate hidden data.
    - **Tool**: **GIMP**, **Photoshop**: Enhance contrast, or zoom in to pixel-level to look for inconsistencies.

Step 3: Extracting Hidden Data from Images

1. **Extract Hidden Data (LSB and Other Formats)**: Use tools like **StegExpose** and **OpenStego** to extract hidden messages or files from images.
    - **Tool**: **OpenStego** can be used to extract the hidden data if it's embedded using OpenStego or other similar tools.

    - `openstego extract -i image.jpg -o extracted_data.txt`
2. **Check for hidden files inside images**: Sometimes, data is embedded within the image, and it can be extracted as a hidden file.
    - **Tool**: **Binwalk** can extract hidden files or data embedded in images.
    - `binwalk -e image.jpg`

4. Detecting Steganography in Audio and Video Files

Audio Steganography (WAV, MP3, etc.)

1. **Analyze audio for hidden data**: Audio files can hide information in the least significant bits of sound waves or use other encoding techniques.
    - **Tool**: **Steghide** supports embedding and extracting data from audio files.
    - `steghide extract -sf audio.wav`
2. **Analyze Frequency and Statistical Anomalies**: Look for irregularities in frequency distribution or waveforms.
    - **Tool**: Use **Audacity** to analyze waveform irregularities.
    - **Tool**: **Sonic Visualiser** for spectral analysis of the audio file.

Video Steganography (AVI, MP4, etc.)

1. **Check for unusual file size**: Similar to images, videos used for steganography may have larger file sizes than expected, even if they appear normal.
2. **Visual Inspection for Anomalies**: Use tools to analyze individual frames and look for noise or subtle changes that could indicate hidden data.
3. **Extraction**: Some tools like **Steghide** and **OpenStego** also support embedding and extraction of hidden data in video files, similar to audio files.

5. Detecting Text-based Steganography

- **Hidden Text in Word Documents or PDFs**: Look for invisible characters or hidden text in documents (e.g., zero-width characters, spaces, line breaks).
- **Tool**: Use **StegExpose** or **Text Steganography Analyzers** to check for invisible characters in text files or documents.
- **Tool**: **StegSolve** can help visualize and detect hidden messages in various formats, including PDFs or text files.

6. File Carving and Data Recovery

Even when steganography is used, remnants of hidden files might still be present in unallocated space on the disk, which can be recovered via **file carving**.

- **Tool**: **Foremost** or **Scalpel** can carve unallocated space to recover potentially hidden files.

7. General Forensic Best Practices for Steganography Detection

1. **Work on forensic copies**: Always work with disk images or write-protected copies to preserve the integrity of the evidence.
2. **Maintain the chain of custody**: Record every action taken, tools used, and files analyzed to ensure proper legal documentation.
3. **Use multiple tools**: Different steganography techniques may require different approaches or tools. Combine the strengths of multiple tools for thorough detection.
4. **Correlate evidence**: If hidden data is found, correlate it with other sources of evidence (e.g., logs, emails) to establish its relevance.

Tools Summary

Tool Name	Description	Use Case
Stegdetect	Detects LSB-based steganography in JPEG images.	Detect hidden data in image files.
StegExpose	Java-based tool for detecting LSB-based steganography.	Detect hidden messages in images.
OpenStego	Open-source tool for detecting/extracting hidden data.	Detect/extract hidden files.
Steghide	Extracts data from audio, image, and text files.	Audio and image file analysis.
Binwalk	Extracts hidden files embedded inside other files.	Carve hidden files from images.
Foremost	File carving tool for recovering hidden or deleted files.	Recover hidden data from unallocated space.

PRACTICAL 4 C.

Aim: Perform a practical of identifying the packer used to pack a file by using ExeInfo PE and then unpacking the file using UPX

Code:

Sure! Here's a step-by-step practical guide for identifying a packer used to pack an executable file and then unpacking it using **UPX** (Ultimate Packer for Executables). This will help you understand how to work with packed files in cyber forensics.

Practical Exercise: Identifying and Unpacking a Packed File

Tools Required:

1. **ExeInfo PE** – Tool for identifying packers and file information of executable files.
2. **UPX (Ultimate Packer for Executables)** – A common tool for packing and unpacking executable files.

Step 1: Identifying the Packer Used on the Executable File Using ExeInfo PE

1. **Download and Install ExeInfo PE**:
   - You can download **ExeInfo PE** from here.
   - Install and open the tool.
2. **Open the Packed Executable File in ExeInfo PE**:
   - Launch **ExeInfo PE**.
   - From the **File menu**, select **Open** and browse to the packed executable file (e.g., `packed_file.exe`).
3. **View Information on the Executable**:
   - Once the file is loaded, ExeInfo PE will display information about the executable.
   - It will automatically try to detect the packer used to pack the file and display the results under the **"Packer/Protector"** section.
4. **Identify the Packer**:
   - Look for the **Packer** or **Protector** field in the information pane.
   - ExeInfo PE will try to recognize if any known packers, like **UPX**, **FSG**, **ASPack**, or **PECompact**, were used.
   - For example, if the file was packed with **UPX**, it will show something like:
     - **UPX 3.x** or similar.
   - If it doesn't detect a known packer, it may just show **"Unknown Packer"** or similar, indicating that it cannot identify the packer.

Step 2: Unpacking the Executable Using UPX

If the file is packed with **UPX** (as identified by ExeInfo PE), you can unpack it easily using **UPX**.

1. **Download and Install UPX**:
    - Download **UPX** from the official website: https://upx.github.io/.
    - You will get a compressed archive that contains the **UPX** executable. Extract it to a folder.
2. **Verify UPX is Working**:
    - Open a **Command Prompt** (or **Terminal** if on Linux/Mac).
    - Navigate to the folder where UPX is extracted.
    - Type `upx` and press **Enter** to check if UPX is recognized. You should see something like:
    - `UPX 3.x.y (Linux/Windows x86-64)`
    - `Copyright (C) 1996-2021 UPX Team`
3. **Unpack the File Using UPX**:
    - Once you confirm UPX is installed, use the following command to unpack the file:
    - `upx -d packed_file.exe`
    - `-d` stands for "decompress," telling UPX to unpack the executable.
4. **Verify the Unpacked File**:
    - After the unpacking process is complete, you should see a message like:
    - `Unpacked 1 file.`
    - The packed file `packed_file.exe` will be unpacked and replaced by the original executable.
5. **Check File Size**:
    - The unpacked file will typically have a **larger file size** than the packed version, since the compression applied by UPX will be removed.

Step 3: Verifying the Unpacked File

1. **Check File Integrity**:
    - To confirm that the unpacked file is functional and not corrupted, you can try running it:
        - Double-click the file to run it (if safe to do so) or analyze it in a controlled, isolated environment like a **sandbox** or **virtual machine**.
2. **Re-run ExeInfo PE**:
    - You can also open the unpacked file in **ExeInfo PE** again to verify that it is no longer packed.

Summary of the Practical Process

1. **Identify the Packer**:
    - Use **ExeInfo PE** to load the packed executable and identify the packing method.
2. **Unpack the Executable**:
    - If the packer is **UPX**, use the `upx -d` command to decompress the executable.
3. **Verify**:
    - Check if the unpacked file runs normally and examine it for any additional changes.

Example Scenario

- **Scenario**: You receive an executable file `example.exe` that you suspect is packed to obfuscate its contents.
    - **Step 1**: Open `example.exe` in **ExeInfo PE**. It reports that the file is packed with **UPX**.
    - **Step 2**: You run the `upx -d example.exe` command to unpack the file.
    - **Step 3**: After unpacking, you verify the file's functionality by executing it in a sandbox or checking it with ExeInfo PE again (which now shows the unpacked file).

Key Takeaways

- **ExeInfo PE** is useful for identifying packers used on executable files.
- **UPX** is a popular packer, and if detected, it can be easily unpacked using the `upx -d` command.
- Forensic practitioners may need to unpack files to analyze them for malicious code, hidden data, or other forensic evidence.

PERFORMING OS FORENSICS

PRACTICAL 5 A.

Aim: Perform a Practical collect volatile information from a host computer running on a Windows OS by using tools PsTools, LogonSessions, and NetworkOpenedFiles

Code:

Certainly! Here's a practical guide for **performing OS forensics** on a Windows host by collecting volatile information using some common forensic tools, namely **PsTools**, **LogonSessions**, and **NetworkOpenedFiles**.

This type of forensic investigation typically involves gathering volatile data that is stored in memory or system resources, which could be cleared upon system shutdown or restart.

Practical: Collect Volatile Information from a Windows Host Using PsTools, LogonSessions, and NetworkOpenedFiles

Tools Required:

1. **PsTools Suite**: A set of command-line utilities from Sysinternals, including **PsExec**, **PsList**, **PsLoggedOn**, etc.
2. **LogonSessions**: A tool from Sysinternals used to gather information about logged-on users.
3. **NetworkOpenedFiles**: A tool to find files that are currently opened over the network on a Windows machine.

Step-by-Step Guide

1. Download and Set Up Tools

1. **Download PsTools Suite**:
   - Download **PsTools** from Microsoft's **Sysinternals** site: PsTools Suite.
   - Extract the suite to a folder on your system.
2. **Download LogonSessions and NetworkOpenedFiles**:
   - Both tools are part of **Sysinternals Suite**.
   - You can also download them individually from the Sysinternals Page.

2. Collect Volatile Information

Step 1: Gathering Information About Logged-on Users

LogonSessions provides information about users who have logged onto the system, their login times, session details, and more.

1. **Launch Command Prompt with Administrator Privileges**:
   - **Right-click** on Command Prompt and select **Run as administrator**.
2. **Navigate to the Directory with Sysinternals Tools**:
   - If you downloaded and extracted **PsTools** and **LogonSessions**, navigate to the folder where these tools are located.
3. **Run LogonSessions**:
   - In the command prompt, type the following command to see who is currently logged on to the machine:
   - `logonsessions.exe`
4. **Interpret Results**:
   - **LogonSessions** will display a list of active user sessions.
     - **Session ID**: Identifies the user's session.
     - **Logon Time**: Time when the user logged into the system.
     - **User Name**: The account name of the user.
     - **Domain**: If applicable, the domain the user is associated with.

   Example output:

```
Logon ID: 0x3e7
User Name: Administrator
Domain: LOCALHOST
Logon Time: 2023-04-25 15:22:32
Session Type: Console

Logon ID: 0x2e5
User Name: User123
Domain: DOMAIN
Logon Time: 2023-04-25 15:23:45
Session Type: RDP
```

   Useful Insight: This will give you information about the logged-on users, which is critical in understanding who was using the system during an investigation.

Step 2: Gathering Information About Running Processes

You can use **PsList** (part of PsTools) to gather information about processes currently running on the system.

1. **Run PsList**:
   - Type the following command in the command prompt to get a list of all active processes:
   - `pslist.exe`
2. **Interpret Results**:
   - **PsList** will show details about each process, such as:
     - **PID (Process ID)**: Unique identifier for the process.
     - **Name**: The name of the process (e.g., `explorer.exe`, `svchost.exe`).

- **Memory Usage**: Amount of system memory used by the process.
- **CPU Usage**: CPU time consumed by the process.
- **Owner**: The user that started the process.

Example output:

```
Name          PID   PPID  Elapsed     CPU  Memory  Description
svchost.exe   4     1000  1d 10h 23m  0.02  1.4MB   Host Process for
Windows Services
explorer.exe  708   760   1d 12h 34m  0.05  10MB    Windows Explorer
chrome.exe    124   708   5h 23m       0.15  400MB   Google Chrome
```

Useful Insight: Analyzing running processes can help you spot any unusual activity, such as processes that are unusual for the system or processes that are using an unusually high amount of resources.

Step 3: Gathering Information About Open Network Files

NetworkOpenedFiles can be used to check which files are currently opened over the network.

1. **Run NetworkOpenedFiles**:
   - Type the following command to get a list of network-shared files that are currently open:
   - `networkopenedfiles.exe`
2. **Interpret Results**:
   - The output will show a list of files opened by the system over the network, including:
     - **File Path**: The full path of the file being accessed.
     - **Accessing User**: The user who has the file open.
     - **Network IP**: The IP address from where the file is being accessed.

Example output:

```
File Path                              User        IP Address
File Opened
\\Server\Share\Document.txt            User123     192.168.1.10
READ
\\Server\Share\FinancialReport.xlsx    Admin       192.168.1.5
WRITE
```

Useful Insight: This can be useful in detecting suspicious or unauthorized file access over the network, especially when files are being accessed from unknown or unauthorized IPs.

Step 4: Additional Volatile Information Collection

To enhance your investigation, you might consider using additional Sysinternals tools:

1. **PsExec**: For remotely executing commands on the host.
    - Example command:
    - `psexec \\hostname -u user -p password cmd.exe`
2. **PsLoggedOn**: To check which users are logged onto a remote machine.
    - Example command:
    - `psloggedon \\hostname`
3. **PsKill**: To terminate suspicious processes on the host.
    - Example command:
    - `pskill.exe pid`
4. **PsFile**: To view which files are open remotely.
    - Example command:
    - `psfile.exe \\hostname`

Step 5: Report and Documentation

1. **Create a Forensic Report**:
    - **Timestamp**: Ensure that you capture the timestamp for each collection step to maintain proper chain of custody.
    - **Tools Used**: List the tools used in the investigation (PsTools, LogonSessions, NetworkOpenedFiles, etc.).
    - **Collected Data**: Document the collected data (processes, logged-on users, network files).
    - **Suspicious Findings**: Highlight any unusual or suspicious findings such as unauthorized users or unknown processes.
2. **Store Data Securely**:
    - Ensure that all collected data is securely stored and that the chain of custody is maintained for all forensic evidence.

PRACTICAL 5 B.

Aim: Perform a Practical for Discovering and Extracting Hidden Forensic Material on Computers Using OSForensics

Code:

Sure! Here's a practical guide on **discovering and extracting hidden forensic material** on computers using **OSForensics**. OSForensics is a comprehensive digital forensics tool that allows you to uncover various types of hidden data on a system, such as deleted files, hidden partitions, and encrypted data.

Practical Exercise: Discovering and Extracting Hidden Forensic Material Using OSForensics

Tools Required:

- **OSForensics**: A powerful tool for forensic investigations that helps identify hidden, deleted, or encrypted data.
- **Computer**: A Windows-based system (either physical or virtual machine) to run OSForensics on.

Step 1: Install and Launch OSForensics

1. **Download OSForensics**:
    - Navigate to the official website of OSForensics: Download OSForensics.
    - Choose the appropriate version (32-bit or 64-bit) for your system.
    - After downloading, run the installer and follow the on-screen instructions to complete the installation.
2. **Launch OSForensics**:
    - Once installed, launch **OSForensics**.
    - You may need administrator privileges to run OSForensics on your computer.

Step 2: Create a Forensic Image (Optional but Recommended)

Before you begin analyzing the target system, it's crucial to ensure that the system's integrity is preserved. One way to do this is by creating a **forensic image** of the system or drive. This ensures that you are working with a copy of the data, rather than directly manipulating the live system.

1. **Create an Image of the Disk**:
    - In OSForensics, go to the **"Tools"** tab and click on **"Create Image"**.
    - Select the **source drive** (e.g., C: drive) that you wish to analyze.

- Specify the **destination folder** where the forensic image will be saved.
- Choose the **image format** (e.g., .E01, .DD, etc.) and click **Start** to create the image.

Step 3: Discovering Hidden Files and Deleted Data

OSForensics can help identify hidden files and recover deleted data, making it an essential tool for forensic investigations.

A. Scan for Deleted Files:

1. **Navigate to "File Carving"**:
   - On the left-hand panel of OSForensics, click on the **"File Carving"** option under the **"Evidence"** section.
2. **Select the Target Disk/Image**:
   - Select the forensic image or the specific drive that you want to search for deleted files.
3. **Run the Carve Process**:
   - Choose the types of files to search for, such as **JPEG**, **PDF**, **DOCX**, etc. You can also select **All File Types** if you want to scan for any kind of file.
   - Start the carving process, which scans for any recoverable deleted files.
4. **Review the Results**:
   - After the carving process is completed, OSForensics will display a list of deleted files that have been recovered.
   - You can preview these files and save them for further analysis.

B. Search for Hidden Files (Unallocated Space):

1. **Go to "Unallocated Space"**:
   - From the **"File Systems"** section in OSForensics, select the **"Unallocated Space"** option. This area often contains fragments of files or hidden data that were once stored on the system.
2. **Run a Search**:
   - Run a search for **specific keywords** or **file signatures** to uncover hidden data. For example, searching for common file extensions like **.jpg**, **.zip**, or **.docx** can help you identify files that are still accessible in the unallocated space but are not part of the file system.
3. **Recover Hidden Files**:
   - If any hidden or fragmented files are found, OSForensics will allow you to recover and save them to another location for further analysis.

C. Scan for Hidden Partitions:

1. **Use the "Partition Table"** Tool:
   - Under the **"Disk Analysis"** section, navigate to **"Partition Table"**.

- OSForensics will display all partitions on the selected disk, including those that may not be visible to the operating system.

2. **Look for Hidden or Unusual Partitions**:
   - Hidden partitions may be used to store illicit data. Review the partition table carefully for any anomalies or partitions that are not commonly seen.
3. **Mount and Access the Hidden Partition**:
   - If a hidden partition is found, you can mount it and analyze its contents using OSForensics.

Step 4: Analyzing and Extracting Network Activity

A. Extracting Network Artifacts:

OSForensics can also help recover and analyze network activity, such as **IP addresses**, **web browsing history**, and **networked file transfers**.

1. **Navigate to "Network"**:
   - Under the **"Internet"** section in OSForensics, you'll find **"Network"** options. Click on **"Network History"**.
2. **Analyze Web Browsing History**:
   - OSForensics will list the **URLs** that have been accessed, the **IP addresses** of remote hosts, and the **timestamps** when these connections occurred.
   - This can provide valuable information about the activity of a user on the system, including potential communications or access to malicious sites.
3. **Examine Network Connections**:
   - You can also check for active or past network connections. This is useful for identifying any **suspicious remote connections** or file-sharing activity.

Step 5: Analyzing System Logs for Hidden Artifacts

System logs can provide critical information about user activities, system events, and even signs of tampering.

1. **Check Event Logs**:
   - In OSForensics, go to **"System Logs"** under the **"Evidence"** section.
   - Review logs such as **Windows Event Logs** and **Application Logs** for any **suspicious activities**.
   - Look for **login attempts**, **system errors**, or **warnings** that could indicate the presence of hidden activities or malware.

Step 6: Recovering Encrypted or Password-Protected Files

If hidden forensic material is encrypted or password-protected, OSForensics provides a way to analyze these files, though cracking passwords can be time-consuming.

A. Identify Encrypted Files:

1. **Use "File Hashing"**:
    - OSForensics can check file hashes to identify files that may have been encrypted or altered.
    - In the **"Evidence"** section, go to **"File Hashing"** and compute hash values for files you suspect are encrypted.
2. **Attempt Password Recovery**:
    - If you suspect a file is password-protected (e.g., a ZIP or RAR file), OSForensics includes a password recovery feature.
    - OSForensics can attempt to **brute-force** passwords for certain file types, though this can take significant time depending on the complexity of the password.

Step 7: Exporting and Reporting the Findings

Once the forensic analysis is complete, OSForensics allows you to export the findings and generate a report for documentation purposes.

1. **Generate a Report**:
    - Go to the **"Report"** tab in OSForensics.
    - Choose the **types of evidence** you want to include (e.g., deleted files, hidden partitions, logs, etc.).
    - Click **"Generate Report"** to create a detailed PDF or HTML report.
2. **Export Files**:
    - You can also manually export any recovered files (e.g., hidden files, deleted files, logs) to a separate folder for further analysis.

Conclusion:

In this practical, you've learned how to use **OSForensics** to discover and extract hidden forensic material on a Windows computer. You've gone through the steps of:

1. **Creating forensic images** for preservation.
2. **Carving deleted files** from unallocated space.
3. **Searching for hidden partitions** and files.
4. **Extracting network activity** and **analyzing system logs**.
5. **Identifying and attempting recovery of encrypted files**.

PRACTICAL 5 C.

Aim: Performing a Computer Forensic Investigation Using the Helix Tool

Code:

Sure! Here's a practical guide to performing a **computer forensic investigation using the Helix tool**. **Helix** is a well-known, live-response forensic tool that allows investigators to gather evidence from a computer without altering the system. It is commonly used for collecting volatile data, investigating memory, disk analysis, and performing live forensics.

Tools Required:

1. **Helix Forensic Tool**: You can download the Helix tool (a live CD or USB) from Helix3.
2. **Target Computer**: A system that will be investigated.
3. **USB Stick or CD/DVD**: To make Helix bootable.
4. **Write Blockers (Optional but recommended)**: To prevent any data modification on the target system during the analysis.

Step 1: Creating a Bootable Helix USB/CD/DVD

1. **Download Helix**:
   - Go to Helix3 and download the latest version of **Helix3**.
2. **Create a Bootable USB or CD/DVD**:
   - If using a **USB**:
     - Use a tool like **Rufus** (available at rufus.ie) to write the Helix image to a USB stick.
     - Insert your USB stick, open Rufus, and select the Helix ISO file to make it bootable.
   - If using a **CD/DVD**:
     - Burn the ISO image to the CD/DVD using a tool like **ImgBurn** or any other ISO burning software.

Step 2: Boot the Target System with Helix

1. **Boot the Target System with Helix**:
   - Insert the **Helix bootable USB** or **CD/DVD** into the target machine.
   - Power on the system and enter the **BIOS/UEFI** settings (usually by pressing **F2**, **Esc**, **Del**, or another key depending on the system).
   - Set the **boot order** so the system boots from the USB or CD/DVD.
   - Save the settings and reboot the system. The computer will now boot into **Helix**.
2. **Helix Live Environment**:
   - Helix will load into a **Linux-based live environment** with a GUI and command-line interface.
   - **Important**: Helix will operate from the bootable media, meaning the target system's hard drive won't be modified, preserving the integrity of the evidence.

Step 3: Collecting Volatile Data

Once Helix has booted into the system, you can start gathering **volatile data** (i.e., data that is lost when the system is powered down), which is crucial for any forensic investigation.

A. Check Running Processes (PsList/Task Manager)

1. **Run Helix's Process Manager**:
   - In the Helix interface, navigate to the **"Process List"** tool.
   - This will provide a live view of the **running processes**, similar to **Task Manager** in Windows.
2. **Review Processes**:
   - Review the list of processes running on the target system. You may want to identify **suspicious processes** or any processes that seem out of place.
   - You can also **terminate suspicious processes** if you are in the middle of an incident response and need to stop malicious activity.

B. Capture System Memory (RAM)

1. **Capture Memory Dump**:
   - Helix includes tools for **capturing system memory** (RAM), which is critical for analysis during forensic investigations.
   - Navigate to **"Memory Dump"** or **"RAM Capture"** under the **Forensic Toolkit** menu.
   - Select the memory capture option and create a **raw dump** of the system's volatile memory.
2. **Why Capture Memory**:
   - The memory dump will contain valuable forensic information such as:
     - **Running processes** and their associated data.
     - **Encryption keys**, if present.
     - **Network connections** and communication data.
     - **Password caches** and other sensitive data.
3. **Save Memory Dump**:
   - Save the memory dump to an external location (e.g., a USB drive) to ensure that it is preserved for further analysis.

C. Check Network Activity

1. **Network Connections**:
   - Helix provides a network tool to view **active network connections**.
   - Navigate to **"Network Connections"** in the Helix interface.
2. **View Active Connections**:
   - Identify any **suspicious remote connections** that could indicate malicious activity.
   - You will be able to see **IP addresses**, **ports**, and **protocols** used.

D. System Information Collection

1. **Collect System Information**:
    - Helix provides a set of tools to gather detailed information about the system's configuration, hardware, and software.
    - Navigate to **"System Information"** to collect:
        - **Hardware details** (CPU, RAM, storage, etc.).
        - **Installed software**.
        - **System logs**.

Step 4: Analyzing File System and Data Extraction

A. Accessing File System

1. **Mounting Local Drives**:
    - Helix can mount the local drive of the target system as a **read-only volume**, ensuring the data isn't altered.
    - Navigate to **"Mount Drives"** and choose the target drive.
2. **Browse the File System**:
    - You can now browse through the files on the target machine without altering the data.
    - Use the **"File Browser"** to search for files, including **hidden, deleted**, or **system files**.

B. Recovering Deleted Files

1. **File Recovery**:
    - Use Helix's built-in **file recovery tool** to search for **deleted files**.
    - Navigate to **"File Recovery"** in the **Forensic Toolkit**.
    - The tool will attempt to recover files from unallocated space on the hard drive.

C. Collecting Evidence

1. **Forensic Copy of Files**:
    - You can create **forensic copies** of important files or folders by right-clicking and selecting **"Copy"**.
    - Make sure the evidence is copied to an external storage device (e.g., USB drive).

Step 5: Documenting and Reporting

As part of your forensic investigation, it is crucial to document your findings and actions taken during the process. This includes:

1. **Creating a Forensic Report**:
    - Document each step of the investigation, including **which tools were used**, **what data was collected**, and **any actions performed**.

    - The report should include:
        - **Case Details**: Description of the incident, date, and time.
        - **Tools Used**: List of tools (Helix, memory dump, file recovery, etc.).
        - **Findings**: Suspicious files, deleted data, and network activity.
2. **Chain of Custody**:
    - Maintain a record of the **chain of custody** to ensure that the evidence remains intact and unaltered.
3. **Save and Secure Evidence**:
    - Securely save any collected evidence (e.g., memory dumps, files, logs) to external media.
    - Consider hashing the files (e.g., SHA256) to ensure that the evidence is not tampered with.

Step 6: Final Steps

After completing the forensic investigation, you can power off the system and, if necessary, perform further analysis using other forensic tools or in a controlled environment like a **virtual machine**.

1. **Safely Remove Media**:
    - Safely eject any external media (USB, external drives) used for storing collected evidence.
2. **Shutdown the Target System**:
    - After gathering all necessary data, you may shut down the system, ensuring the integrity of the evidence is maintained.

Conclusion

In this practical guide, we've used **Helix** to perform a live forensic investigation. Here's a summary of the main steps:

- **Booting Helix** on the target machine.
- **Collecting volatile data**, such as RAM dumps, running processes, and network activity.
- **Recovering deleted files** and analyzing the file system.
- **Creating forensic reports** to document the investigation process.

By using Helix, investigators can perform non-intrusive investigations on live systems, ensuring evidence is preserved without altering the state of the target machine.

Would you like additional details on specific Helix tools or steps, or need guidance on interpreting forensic findings?

PRACTICAL 5 D.

Aim: Examine Windows event logs using Event Log Explorer

Code:

Examining Windows Event Logs using Event Log Explorer is a critical part of any digital forensic investigation. Event logs provide detailed records of system activities, such as user logins, application crashes, system events, security alerts, and more. By analyzing event logs, investigators can reconstruct the timeline of actions on a system, detect anomalies, and uncover potential malicious activities.

Event Log Explorer is a specialized tool that helps with reading, searching, and analyzing Windows Event Logs. It is designed to handle large log files, making it more efficient than using the built-in Windows Event Viewer.

Tools Required:

- **Event Log Explorer**: A third-party software used to view and analyze Windows Event Logs.
  - Download it from Event Log Explorer.
- **Target System**: A system with Windows Event Logs (this can be a live system or an image of a system from which logs need to be analyzed).

Step 1: Install and Launch Event Log Explorer

1. **Download Event Log Explorer**:
   - Visit the Event Log Explorer website.
   - Download the version compatible with your Windows OS.
2. **Install Event Log Explorer**:
   - Run the installer and follow the on-screen instructions to complete the installation.
   - Once installed, launch **Event Log Explorer**.

Step 2: Opening Windows Event Logs in Event Log Explorer

Event Log Explorer allows you to open and analyze Windows Event Logs. The most common log files you will work with are the **System**, **Security**, and **Application** logs. Here's how to open them in Event Log Explorer:

1. **Select the Event Log Source**:
   - After launching Event Log Explorer, you will be presented with the **"Select Log File"** dialog box.
   - You can open logs from a **local machine**, **remote machine**, or an **event log file** (in .EVT or .EVTX format).

        - To open local machine logs, select **"Local machine"** and choose the log type (Application, Security, System).
        - To open a remote machine's logs, select **"Remote machine"**, enter the machine name, and provide appropriate credentials.
2. **Choose the Log Type**:
    - You'll generally work with three main types of event logs:
        - **Application Logs**: Logs related to applications and software running on the system.
        - **Security Logs**: Logs related to login attempts, access control, auditing, and other security events.
        - **System Logs**: Logs for operating system events, including system crashes, device failures, and other critical events.
3. **Open the Log**:
    - Click on **"Open"** to open the selected log file(s).

Step 3: Navigating and Analyzing the Logs

Once the log is open in Event Log Explorer, you will see a detailed view of all events stored in the log file.

A. Filtering Events

Event logs can contain thousands of entries, so filtering events to focus on specific activities is essential.

1. **Filter by Event ID**:
    - Event IDs are numeric codes associated with each type of event. You can filter logs by specific **Event IDs** to look for common types of events, such as login attempts, privilege escalation, or system errors.
    - For example:
        - **Event ID 4624**: Successful user login (Windows Security Log).
        - **Event ID 4634**: User logoff (Windows Security Log).
        - **Event ID 6005**: System startup (System Log).
        - **Event ID 6006**: System shutdown (System Log).
2. **Filter by Date/Time**:
    - Use the **time filters** to focus on a specific date and time range when an incident might have occurred.
3. **Filter by Event Level**:
    - Event logs have various **event levels**: **Information**, **Warning**, **Error**, and **Critical**. You can filter by event level to narrow down the scope.
        - For example, filtering to only **Critical** or **Error** events might help identify system failures or signs of an attack.
4. **Search for Keywords**:
    - You can use the **Search** function (Ctrl+F) to find specific keywords, such as the name of an application, user, or IP address.

B. Viewing Event Details

Each event has associated details, including:

- **Event ID**: Numeric code identifying the event type.
- **Date and Time**: When the event occurred.
- **Source**: The component that generated the event (e.g., Windows security, application, system).
- **Event Level**: The severity of the event (Information, Warning, Error).
- **User**: The user account associated with the event (if applicable).
- **Description**: A detailed explanation of the event.

To view the full details of an event, simply click on it, and the details will appear at the bottom of the window.

C. Key Events to Look For

- **Logins and Logoffs (Security Log)**:
  - **Event ID 4624**: Successful login event.
  - **Event ID 4634**: Logoff event.
  - **Event ID 4647**: User-initiated logoff.
- **Failed Logins (Security Log)**:
  - **Event ID 4625**: Failed login attempts.
  - **Event ID 4771**: Kerberos authentication failures.
- **System Events (System Log)**:
  - **Event ID 6005**: Event indicating system startup.
  - **Event ID 6006**: System shutdown event.
  - **Event ID 41**: Unexpected shutdown event (this is useful for detecting sudden power loss or crashes).
- **Security-Related Events (Security Log)**:
  - **Event ID 4720**: User account creation.
  - **Event ID 4726**: User account deletion.
  - **Event ID 4672**: Special privileges assigned to new logon (this may indicate a user was granted administrative privileges).
- **Application Events (Application Log)**:
  - Look for any **warnings** or **errors** related to critical applications.
  - Pay attention to event descriptions that might point to suspicious activity, such as unauthorized application execution or unexpected behavior from trusted applications.

Step 4: Exporting and Saving Logs

Event Log Explorer allows you to export logs for further analysis or reporting.

1. **Export Logs**:
   - To export the events, go to **File** > **Export**.

   - Choose the file format (e.g., CSV, TXT, or HTML) and the location where you want to save the file.
2. **Save Filtered Logs**:
   - If you've applied filters to narrow down the logs, you can export the filtered results for use in a forensic report or further analysis.

Step 5: Creating a Forensic Report

After analyzing the logs, it's important to document your findings in a detailed **forensic report**.

1. **Document Key Findings**:
   - **Logins/Logoffs**: Detail any suspicious logins or logoff events, especially those occurring outside of normal working hours or involving unusual user accounts.
   - **Failed Login Attempts**: Investigate any repeated failed login attempts that might indicate brute force attacks.
   - **Critical System Events**: Look for system crashes, unexpected shutdowns, or errors that may point to compromised system stability.
2. **Generate a Report**:
   - Use the **Export** function to generate a report of your findings.
   - The report should include:
     - **Description of the Events**: Summarize the key events, their IDs, and their significance.
     - **Evidence and Screenshots**: Include screenshots or logs of the relevant events to support your analysis.
     - **Recommendations**: Based on the findings, make recommendations for any corrective actions or additional investigation.

Step 6: Using Event Log Explorer for Ongoing Monitoring (Optional)

Event Log Explorer can also be used for **continuous monitoring** of Windows event logs, allowing you to detect suspicious activities in real time.

1. **Set up Alerts**:
   - Event Log Explorer can be configured to send **alerts** based on specific events, such as login failures or application crashes.
   - You can define custom filters and set up the system to notify you of unusual or potentially malicious behavior.
2. **Monitor Remote Systems**:
   - If you're managing a network of systems, you can use Event Log Explorer to monitor event logs from multiple machines remotely, providing a centralized view of security and system health.

Conclusion

Event Log Explorer is an excellent tool for examining and analyzing Windows event logs. By following these steps, you can efficiently:

- **Identify and filter specific events** based on your investigation goals.
- **Examine security, system, and application logs** to uncover suspicious activities and trace user actions.
- **Export logs** and create forensic reports that detail your findings.

By understanding and analyzing Windows Event Logs, you can gain valuable insight into a system's history, uncover potential security breaches, and piece together critical evidence for forensic investigations.

Would you like additional help with specific types of event logs, or need assistance with interpreting log data?

NETWORK FORENSICS

PRACTICAL 6 A.

Aim: Investigating Network Traffic Using Wireshark

Code:

Step 1: Installing Wireshark

1. **Download Wireshark**:
   - Go to the official Wireshark website: https://www.wireshark.org/download.html.
   - Select the appropriate version for your operating system (Windows, macOS, Linux).
2. **Install Wireshark**:
   - Follow the installation prompts to install Wireshark on your system.
   - Ensure that you also install **WinPcap** (for Windows) or **libpcap** (for macOS/Linux), as they are required to capture live network traffic.

Step 2: Capturing Network Traffic

1. **Launch Wireshark**:
   - Open Wireshark after installation.
2. **Select Network Interface**:
   - In the Wireshark interface, you'll see a list of available **network interfaces** (such as Ethernet, Wi-Fi, etc.).
   - Select the interface you want to monitor. For example, if you're monitoring wireless traffic, select the **Wi-Fi interface**.
   - Click on the interface to start capturing traffic.
3. **Start Capturing**:
   - Once you select the interface, Wireshark will start capturing packets in real-time.
   - You'll see a list of captured packets, which includes information such as the **packet number**, **timestamp**, **source IP**, **destination IP**, **protocol**, and more.
4. **Stop Capturing**:
   - Click the red **square button** at the top left of the Wireshark window to stop the capture when you've gathered enough data.

Step 3: Analyzing Network Traffic

Once you've captured some traffic, you can start analyzing it to look for suspicious activities or specific events of interest.

A. Filtering Traffic

Wireshark allows you to **filter traffic** using display filters. Some commonly used filters include:

- **IP Traffic**:

    - To filter packets from a specific source IP:
    - `ip.src == 192.168.1.5`
    - To filter packets going to a specific destination IP:
    - `ip.dst == 192.168.1.10`
- **Protocol Filtering**:
    - To filter for **TCP traffic**:
    - `tcp`
    - To filter for **UDP traffic**:
    - `udp`
- **HTTP Traffic**:
    - To view only **HTTP traffic**:
    - `http`
- **DNS Traffic**:
    - To view **DNS requests**:
    - `dns`
- **Follow TCP Stream**:
    - Right-click on any TCP packet and select **"Follow" > "TCP Stream"** to view the full conversation between the client and server.

B. Identifying Suspicious Activities

Look for anomalies in the traffic such as:

1. **Large Data Transfers**:
    - Excessive data transfers may indicate data exfiltration.
2. **Unusual Ports**:
    - If you observe traffic on **non-standard ports** (other than ports 80 for HTTP, 443 for HTTPS), it could be a sign of malicious activities.
3. **Failed Login Attempts**:
    - Multiple failed login attempts, especially from a single source IP, could indicate a brute-force attack.
4. **Suspicious Protocols**:
    - Certain protocols, such as **NetBIOS**, **SMB**, or **Telnet**, are sometimes exploited by attackers. Look for these in unexpected places.

C. Exporting and Saving Captured Packets

1. **Export Data**:
    - After capturing and analyzing the traffic, you may want to **save the packet data** for future analysis or reporting.
    - Go to **File** > **Save As** to save the capture file (typically with a `.pcap` or `.pcapng` extension).
2. **Export Specific Packets**:
    - You can also export **filtered packets** by selecting **File** > **Export Specified Packets**. This allows you to save only the relevant packets you've filtered.

Step 4: Using Wireshark for Advanced Analysis

Wireshark provides several advanced features that allow you to drill deeper into the packet data. Some useful techniques include:

1. **Packet Analysis with Color Coding**:
   - Wireshark uses color coding to help identify different protocols and issues at a glance. For example, **TCP Retransmissions** are colored red to alert you to potential network issues.
2. **TCP/UDP Stream Analysis**:
   - You can analyze a full **TCP or UDP stream** to follow the communication between the client and server.
   - Right-click on a TCP or UDP packet, then select **Follow** > **TCP Stream** or **Follow** > **UDP Stream**.
3. **Statistics and Graphs**:
   - Wireshark has several built-in **statistical tools** to help you visualize the network traffic:
     - **Protocol Hierarchy**: To see a breakdown of protocols.
     - **IO Graphs**: To visualize traffic over time.
     - **Endpoints**: To view the source and destination IPs and analyze network traffic patterns.
4. **Decryption of Encrypted Traffic (Optional)**:
   - If you have access to the necessary encryption keys (e.g., SSL/TLS keys), Wireshark can decrypt **HTTPS traffic**.
   - Under **Preferences** > **Protocols** > **TLS**, you can enter the decryption keys.

Step 5: Example Code for Basic Traffic Analysis in Python using Pyshark

If you want to automate traffic analysis, you can use **Pyshark**, a Python wrapper for **Wireshark** (tshark). Here's a simple example of how to use **Pyshark** for network traffic analysis.

Install Pyshark:

```
pip install pyshark
```

Example Code: Analyzing Packets with Pyshark

```
import pyshark

# Define the capture file or live interface (e.g., eth0 or en0 for Wi-Fi)
capture = pyshark.FileCapture('example.pcap')  # Or use 'interface' for live
capture

# Loop through packets in the capture file
for packet in capture:
    # Print out general info about each packet
    print(f"Packet Number: {packet.number}")
    print(f"Timestamp: {packet.sniff_time}")
```

```
    # Print IP info
    if 'IP' in packet:
        print(f"Source IP: {packet.ip.src}")
        print(f"Destination IP: {packet.ip.dst}")

    # Print TCP info (if available)
    if 'TCP' in packet:
        print(f"Source Port: {packet.tcp.srcport}")
        print(f"Destination Port: {packet.tcp.dstport}")

    # Print HTTP info (if available)
    if 'HTTP' in packet:
        print(f"HTTP Request: {packet.http.request_method}
{packet.http.host}")

    print("-" * 50)
```

Step 6: Reporting

When investigating network traffic, you'll need to generate a report of your findings. Your report should include:

1. **Summary of Suspicious Activity**:
    - Describe the types of suspicious activity observed in the network traffic, such as unauthorized access, port scanning, or data exfiltration attempts.
2. **Key Events**:
    - Highlight important network events such as failed login attempts, unusual traffic patterns, or communication with known malicious IP addresses.
3. **Suggested Mitigation**:
    - Provide suggestions for mitigating any threats discovered during the investigation, such as blocking specific IPs, updating firewalls, or enforcing stronger encryption.
4. **Supporting Evidence**:
    - Include relevant **Wireshark capture files** (.pcap) and any code used in the analysis.

Conclusion

Wireshark is an invaluable tool for analyzing network traffic and uncovering signs of malicious activity. By capturing network packets, applying filters, and analyzing the data in real-time, you can gain insights into potential security threats and network anomalies. For automation, you can leverage **Pyshark** to create Python scripts for basic traffic analysis.

Would you like to go deeper into a specific area of network forensics using Wireshark, or need more examples of analysis techniques?

PRACTICAL 6 B.

Aim: Investigating Network Attacks using Kiwi Log Viewer

Code:

Step 1: Installing Kiwi Log Viewer

Kiwi Log Viewer is a product of **SolarWinds**, and it can be used to view, analyze, and search through log files (such as **Syslog**, **SNMP traps**, **Windows event logs**, etc.).

1. **Download Kiwi Log Viewer**:
   - Visit the SolarWinds Kiwi Log Viewer website.
   - Download the free version or trial version of **Kiwi Log Viewer**.
2. **Install Kiwi Log Viewer**:
   - Run the downloaded installer and follow the installation steps.

Step 2: Setting Up Kiwi Log Viewer

1. **Launch Kiwi Log Viewer**:
   - After installation, open **Kiwi Log Viewer**.
2. **Configure Log Sources**:
   - If you're analyzing logs from devices (such as firewalls, routers, or servers), configure them to send **Syslog** messages to your Kiwi Log Viewer.
   - For **Windows logs**, you can use the **Kiwi Syslog Server** or configure Windows Event Logs to send to Kiwi Log Viewer.
3. **Specify Log Files to Monitor**:
   - You can specify particular log files to monitor. These could include files like:
     - **Syslog Logs**: Logs of network events from routers/firewalls.
     - **Windows Event Logs**: Security and application logs from Windows machines.
     - **Security Logs**: Any network or system security events (failed logins, unauthorized access, etc.).

Step 3: Investigating Network Attacks Using Kiwi Log Viewer

Once Kiwi Log Viewer is set up and receiving logs from the system or network devices, you can start investigating potential network attacks. Some common network attack patterns you may want to look for include:

A. Detecting Brute Force Attacks

Brute force attacks usually involve multiple failed login attempts in a short period of time. Here's how you can investigate failed login attempts:

1. **Filter Logs for Failed Logins**:

- Use the **Filter** function in Kiwi Log Viewer to filter out **failed login attempts**. This could be based on:
  - Specific event IDs (e.g., in **Windows Event Logs**, Event ID **4625** corresponds to a failed login).
  - Searching for keywords such as **"login failed"**, **"authentication failure"**, or similar.

Example filter for Windows Event Logs:

```
Event ID: 4625
```

2. **Check Source IP for Multiple Attempts**:
   - Identify if multiple failed login attempts are coming from the same **source IP**.
   - You can then determine whether this is an **IP address** associated with an attack (e.g., a brute-force attempt).

B. Detecting DDoS Attacks

DDoS (Distributed Denial of Service) attacks are characterized by an overwhelming amount of traffic sent to a server or network. Signs of a DDoS attack might include:

1. **Look for Excessive Traffic in Logs**:
   - Filter the logs for unusually high traffic. This could be identified by looking for:
     - **IP address flooding**: A large number of packets from one or more source IPs.
     - **Protocol Anomalies**: Multiple requests using the same protocol (HTTP, ICMP) in rapid succession.
2. **Examine IP Source and Destination Information**:
   - Look for **repeated requests** from the same IP or **multiple requests** from different IPs targeting the same server or service.

Example filter:

```
Source IP: 192.168.1.1
```

C. Detecting Suspicious Port Scanning

Port scanning is often used by attackers to identify open ports and vulnerabilities. Look for repeated connection attempts to various ports:

1. **Examine Logs for Excessive Connections to Different Ports**:
   - Filter logs for **multiple connection attempts** to different ports from a single IP.
2. **Suspicious Protocols**:
   - Look for protocols like **TCP SYN packets** or **ICMP echo requests** which are commonly used in port scanning.

Example filter for **TCP SYN** packets:

```
    Protocol: TCP, Type: SYN
```

D. Malicious Traffic Patterns

Look for signs of malicious or suspicious traffic, such as:

1. **Unusual Traffic Between Known Hosts**:
   - Look for **traffic between internal systems** that typically shouldn't communicate.
2. **Connection to Known Malicious IPs**:
   - Filter for **outbound connections** to IPs known to be associated with malicious activities (this can be cross-referenced with threat intelligence sources).
3. **Unusual Network Activity on Ports**:
   - Monitor for unusual network activity on **non-standard ports** (ports other than 80, 443, 21, etc.).

Step 4: Automating Log Analysis Using Python

You can automate the analysis of network logs (for example, Syslog logs, Windows Event Logs) using Python. The **PyKiwi** package can be used to interact with Kiwi Syslog and automate tasks like log parsing and filtering. If you're analyzing log files that you've already exported from Kiwi Log Viewer, here's an example using **Python**.

Install Required Libraries

```
pip install pywin32
pip install pandas
```

Example Python Code: Parsing and Analyzing Logs

The following Python script will parse logs and look for specific patterns such as failed login attempts and suspicious IPs.

```
import re
import pandas as pd

# Path to the exported log file from Kiwi Log Viewer (e.g., .txt or .csv
format)
log_file_path = "exported_log_file.txt"

# Read the log file
with open(log_file_path, 'r') as file:
    logs = file.readlines()

# Define a function to extract relevant details from the logs
def extract_failed_logins(log_lines):
    failed_login_pattern = r"Login failed from IP: (\d+\.\d+\.\d+\.\d+)"
    failed_logins = []

    for line in log_lines:
        # Look for failed login attempts based on the regex pattern
```

```
        match = re.search(failed_login_pattern, line)
        if match:
            failed_logins.append(match.group(1))  # Store the source IP of
failed logins

    return failed_logins

# Extract failed login attempts
failed_logins = extract_failed_logins(logs)

# Analyze failed login IPs (Detect IPs that appear more than 3 times,
indicating a brute force attack)
login_counts = pd.Series(failed_logins).value_counts()

# Filter IPs with more than 3 failed login attempts
suspicious_ips = login_counts[login_counts > 3]
print("Suspicious IPs (Potential Brute Force Attacks):")
print(suspicious_ips)

# Example: Checking for excessive traffic patterns (e.g., more than 10
connections to different ports)
def check_excessive_traffic(log_lines):
    ip_traffic = {}

    for line in log_lines:
        # Look for connection attempts based on the pattern (example:
"Connection from IP: x.x.x.x")
        match = re.search(r"Connection from IP: (\d+\.\d+\.\d+\.\d+)", line)
        if match:
            ip = match.group(1)
            ip_traffic[ip] = ip_traffic.get(ip, 0) + 1

    # Filter IPs with more than 10 connections
    suspicious_traffic = {ip: count for ip, count in ip_traffic.items() if
count > 10}

    return suspicious_traffic

# Check for excessive traffic
excessive_traffic = check_excessive_traffic(logs)
print("\nExcessive Traffic Detected from IPs:")
print(excessive_traffic)
```

Explanation of Code:

1. **Reading Logs**:
    - The log file is read line by line using Python's `open()` function.
2. **Pattern Matching (Regex)**:
    - The script uses **regular expressions** (`re.search()`) to match patterns like failed login attempts or IP addresses in the logs.
3. **Pandas for Data Analysis**:
    - **Pandas** is used to aggregate the count of occurrences of IP addresses. This helps to easily detect repeated failed login attempts or suspicious traffic.

4. **Suspicious Activity Detection**:
    - The script identifies IPs that have failed login attempts more than 3 times (potential brute force attacks).
    - It also checks for **excessive traffic** by counting connection attempts from the same IP address.

Step 5: Reporting and Next Steps

Once suspicious activities are identified, you can create a report to document the findings, which should include:

1. **Summary of Suspicious Activity**:
    - Include details of brute force attempts, DDoS patterns, port scans, and any other malicious activities detected in the logs.
2. **IP Addresses**:
    - Highlight the suspicious IPs, especially those with excessive failed login attempts or excessive connections to different ports.
3. **Recommended Actions**:
    - Suggest mitigation steps like blocking suspicious IPs, increasing authentication

INVESTIGATING WEB ATTACKS

PRACTICAL 7.

Aim: Analyzing Domain and IP Address Queries Using SmartWhois Tool

Code:

Investigating Web Attacks: Analyzing Domain and IP Address Queries Using the SmartWhois Tool

SmartWhois is a tool that allows users to query domain names, IP addresses, and WHOIS databases to retrieve information about domain owners, IP address locations, and other important details. It can be invaluable for investigating web attacks by tracking malicious actors through domain name registration and IP address details.

Step 1: Downloading and Installing SmartWhois

1. **Download SmartWhois**:
   - Visit the SmartWhois download page to download the tool for your operating system (Windows, macOS, Linux).
2. **Install SmartWhois**:
   - Follow the installation steps to set up SmartWhois on your machine.
3. **Launch SmartWhois**:
   - After installation, open SmartWhois.

Step 2: Analyzing Domain Information with SmartWhois

SmartWhois allows you to query **domain names** and **IP addresses** to retrieve detailed information about the owner, registrar, and other registration details. To analyze web attacks, you can query:

1. **Domain Name**:
   - This can give you information about the website's owner, creation and expiration dates, name servers, and other details.
2. **IP Address**:
   - Querying an IP address will provide information about its location, ISP, and potential associations with malicious activity.

A. Querying Domain Information

To perform a **domain query**:

1. Open SmartWhois.
2. In the "Query" field, type the domain name (e.g., `example.com`).
3. Click the **"Whois"** button to retrieve details about the domain.

SmartWhois Output for Domain Query:

- **Domain Name**: example.com
- **Registrar**: GoDaddy, Inc.
- **Registrant**: John Doe (or anonymous)
- **Creation Date**: 2000-01-01
- **Expiration Date**: 2024-01-01
- **Name Servers**: ns1.example.com, ns2.example.com
- **Registrant Email**: johndoe@example.com (or redacted)

This information can help investigators identify if a malicious domain was registered recently, if it has been used for phishing, or if it is associated with other suspicious domains.

B. Querying IP Address Information

To perform an **IP address query**:

1. In the "Query" field of SmartWhois, type the **IP address** (e.g., `192.168.1.1`).
2. Click the **"Whois"** button to retrieve details about the IP address.

SmartWhois Output for IP Query:

- **IP Address**: 192.168.1.1
- **Location**: United States, New York
- **ISP**: AT&T
- **Network Provider**: AT&T
- **Domain Names Hosted on This IP**: example.com, testsite.com
- **Abuse Contact**: abuse@att.net

This information can help identify if an IP address is associated with a known malicious provider or if multiple domains are hosted on the same IP address that could indicate a compromised server.

Step 3: Using SmartWhois for Investigating Web Attacks

Here's how SmartWhois can assist in investigating potential web attacks:

1. **Phishing Attacks**:
    - If you are investigating a potential phishing attack, you can use SmartWhois to check the domain registration details. Look for:
        - **Recently created domains**: Phishing sites are often set up quickly and may have short expiration times.
        - **Anonymous registrants**: Some phishing domains might have hidden registration details or fake contact information.
2. **DDoS (Distributed Denial of Service) Attacks**:
    - Use SmartWhois to query **IP addresses** that are suspected of being part of a DDoS attack.

        - Look for multiple malicious domains or servers hosted on the same IP address.
3. **Malware Hosting**:
    - Investigate **IP addresses** and **domains** that are suspected of hosting malicious content or malware. Look for known **malicious IP ranges** or suspicious **domain owners**.
4. **Spam and Botnets**:
    - Query domains and IP addresses linked to spam emails or botnet command and control (C&C) servers.

Step 4: Automating Domain and IP Analysis Using Python

You can automate the process of querying domains and IP addresses using the **`whois`** Python module, which allows you to perform WHOIS queries from your Python scripts.

Install Python WHOIS Module

```
pip install python-whois
```

Python Code for Automating Domain and IP WHOIS Queries

The following Python script demonstrates how to perform WHOIS queries for a domain and IP address and parse the results for analysis.

```
import whois
import socket
import requests

# Function to query domain information
def get_domain_info(domain):
    print(f"Querying WHOIS information for domain: {domain}")
    w = whois.whois(domain)

    domain_info = {
        'Domain Name': w.domain_name,
        'Registrar': w.registrar,
        'Creation Date': w.creation_date,
        'Expiration Date': w.expiration_date,
        'Registrant': w.registrant_name,
        'Name Servers': w.name_servers,
        'Emails': w.emails
    }

    return domain_info

# Function to query IP address information using the IPinfo.io API
def get_ip_info(ip_address):
    print(f"Querying IP information for IP: {ip_address}")

    # Using IPinfo.io API to retrieve IP information
```

```
    access_token = 'your_ipinfo_access_token'  # You can get an access token
from https://ipinfo.io/signup
    url = f'https://ipinfo.io/{ip_address}/json?token={access_token}'
    response = requests.get(url)

    ip_info = response.json()
    return ip_info

# Example usage of the functions
if __name__ == "__main__":
    # Example domain and IP address for queries
    domain = 'example.com'
    ip_address = '8.8.8.8'

    # Get domain information
    domain_info = get_domain_info(domain)
    print("\nDomain Info:")
    for key, value in domain_info.items():
        print(f"{key}: {value}")

    print("\n" + "-"*50)

    # Get IP address information
    ip_info = get_ip_info(ip_address)
    print("\nIP Info:")
    for key, value in ip_info.items():
        print(f"{key}: {value}")
```

Explanation of Code:

1. **`get_domain_info(domain)`**:
   - This function queries the **WHOIS** information for a given domain using the `whois` module.
   - It returns important details like the domain name, registrar, creation and expiration dates, name servers, and email contacts.
2. **`get_ip_info(ip_address)`**:
   - This function queries the **IP address information** using the **IPinfo.io API**.
   - It returns details like the location of the IP address, the ISP, and any associated domains.
   - You'll need to sign up for an API key from IPinfo.io to use this function.

Step 5: Example Output

Here's an example of the output when querying domain and IP information:

Domain Info (Example.com):

```
Querying WHOIS information for domain: example.com

Domain Name: example.com
Registrar: GoDaddy, Inc.
Creation Date: 1995-01-01
```

```
Expiration Date: 2024-01-01
Registrant: John Doe
Name Servers: ns1.example.com, ns2.example.com
Emails: johndoe@example.com
------------------------------------------------------------
```

IP Info (8.8.8.8 - Google DNS):

```
Querying IP information for IP: 8.8.8.8

IP Address: 8.8.8.8
Location: Mountain View, California, United States
ISP: Google LLC
Hostname: google-public-dns-a.google.com
Org: Google LLC
Country: US
Region: California
City: Mountain View
------------------------------------------------------------
```

Conclusion

By using **SmartWhois** and automating domain and IP address analysis through Python, investigators can gather valuable information for tracking down malicious actors involved in web attacks. Whether investigating phishing attacks, DDoS activities, malware hosting, or other types of web-based threats, these tools allow you to quickly retrieve information about domain registration details and IP address locations to assist in your investigation.

Would you like to explore other tools or dive deeper into automated analysis with more advanced Python techniques? Let me know!

DATABASE FORENSICS

PRACTICAL 8.

Aim: Analyzing SQLite Databases using DB Browser for SQLite

Code:

Database Forensics: Analyzing SQLite Databases using DB Browser for SQLite

SQLite databases are widely used in applications ranging from mobile devices to desktop software and web applications. When performing database forensics, investigators often need to analyze SQLite files to extract valuable information such as user credentials, logs, and other sensitive data.

Step-by-Step Process Using DB Browser for SQLite

Step 1: Opening the SQLite Database

1. **Launch DB Browser for SQLite**:
   - Open **DB Browser for SQLite** on your machine.
2. **Open Database**:
   - In the **DB Browser for SQLite**, click on **"Open Database"**.
   - Select the SQLite file (e.g., `application.db`) you want to analyze.

Step 2: Inspect Database Structure

Once the database is loaded, inspect the structure of the database to understand its contents.

1. **View Database Structure**:
   - In the **Database Structure** tab, you will see a list of tables such as `users`, `logins`, and `transactions`.
2. **Examine Tables**:
   - Click on each table to view the columns it contains:
     - **users table**: Columns may include `id`, `username`, `email`, `password_hash`.
     - **logins table**: Columns may include `user_id`, `login_time`, `ip_address`.
     - **transactions table**: Columns may include `transaction_id`, `user_id`, `amount`, `timestamp`.

Step 3: Query Data

Once you have identified the tables, you can query them to extract useful forensic data using the **Execute SQL** tab.

1. **Get All Users**:
   - To see all users in the database, use the following SQL query:
   - `SELECT * FROM users;`
2. **Filter Users by Email**:

- If you suspect that a particular user is involved in suspicious activities, you can filter by email:
- `SELECT * FROM users WHERE email = 'suspicioususer@example.com';`

3. **View User Login Attempts**:
   - To view the login attempts of a specific user, you can query the `logins` table. For example, to get all login attempts for the user with `user_id = 1`:
   - `SELECT * FROM logins WHERE user_id = 1;`
4. **Get All Failed Logins**:
   - If you have a failed login flag or you know failed login attempts have a specific pattern in the `logins` table, you can filter for those:
   - `SELECT * FROM logins WHERE login_status = 'failed';`
5. **Investigate Transactions for Suspicious Activity**:
   - To look at transactions made by a particular user, you can query the `transactions` table:
   - `SELECT * FROM transactions WHERE user_id = 1;`
6. **Filter Transactions by Date or Amount**:
   - To see transactions above a certain amount or within a specific time range, you can use:
   - `SELECT * FROM transactions WHERE amount > 1000;`

     Or to filter by date:

     ```
     SELECT * FROM transactions WHERE timestamp BETWEEN '2023-01-01'
     AND '2023-01-31';
     ```

Step 4: Analyzing User Activity and Identifying Suspicious Behavior

Let's assume you want to find suspicious activity related to **failed login attempts** and **transactions**:

1. **Query Failed Logins by User**:
   - To see how many failed login attempts were made by each user, you can group the results by `user_id`:
   - `SELECT user_id, COUNT(*) AS failed_logins`
   - `FROM logins`
   - `WHERE login_status = 'failed'`
   - `GROUP BY user_id`
   - `ORDER BY failed_logins DESC;`
2. **Check for Unusual Transaction Behavior**:
   - You might want to find users with the highest value transactions (possible fraud indicators). Query transactions above a specific threshold:
   - `SELECT user_id, SUM(amount) AS total_spent`
   - `FROM transactions`
   - `GROUP BY user_id`
   - `HAVING total_spent > 5000;`
3. **Correlate Logins with Transactions**:
   - If you are looking for transactions made by users who logged in at a specific time (e.g., at night), you can join the `logins` and `transactions` tables:

```
SELECT t.user_id, t.transaction_id, t.amount, t.timestamp,
l.login_time
FROM transactions t
JOIN logins l ON t.user_id = l.user_id
WHERE l.login_time BETWEEN '2023-01-01 00:00:00' AND '2023-01-01
06:00:00';
```

Step 5: Exporting Data

If you need to save the query results for further analysis or documentation, you can export the data.

1. **Export Data**:
   - After running a query, you can export the results to formats such as **CSV** or **SQL**.
   - To export the data, click on **File > Export** and choose the desired export format.

Step 6: Examining Deleted Data (Advanced)

SQLite databases support **journal files** and **undo logs**, which can sometimes allow recovery of deleted records. However, DB Browser for SQLite doesn't directly allow you to view deleted data, but if you are looking for **deleted records**, you may need to:

1. **Look for Database Journal Files**:
   - If you have access to the journal file (e.g., `application.db-journal`), it may contain deleted records or recent changes. You could attempt to use special recovery tools or queries to recover deleted data.
2. **Check for Logical Deletion Flags**:
   - Some applications implement **soft deletion** where deleted records are not physically removed but instead marked as deleted by setting a `deleted` flag to `1`. Check for such flags in tables:
   - `SELECT * FROM users WHERE deleted = 1;`

Step 7: Generating Reports and Final Analysis

After extracting and analyzing data, it's often helpful to summarize findings in a report for legal or investigative purposes.

- You can summarize suspicious activity, unusual transactions, or login patterns based on your queries.
- You can generate reports based on your query results in **CSV** or **SQL** formats.

Conclusion

By using **DB Browser for SQLite** and SQL queries, you can effectively analyze SQLite databases during a forensic investigation.

MALWARE FORENSICS

PRACTICAL 9 A.

Aim: Perform Static Analysis of the Suspicious File

Code:

Malware Forensics: Static Analysis of a Suspicious File

Static analysis is a process where a suspicious file is analyzed without being executed. In malware forensics, static analysis helps determine the characteristics of a suspicious file, such as its structure, embedded strings, libraries used, and any other patterns that might suggest malicious intent. Static analysis is typically the first step in analyzing a suspicious file, followed by dynamic analysis (executing the file in a controlled environment).

Step-by-Step Static Analysis of a Suspicious File

Let's assume we have a suspicious file named `malicious_sample.exe` that you need to analyze. We'll walk through the static analysis process with a few steps.

Step 1: File Identification

The first step in static analysis is identifying the type and format of the file.

1. **Check File Type and Format**:
    - Use the `file` command (Linux/macOS) or **TrID** (Windows) to determine the file type. This helps identify whether the file is an executable, a script, or a document.
    - Example command on Linux:
    - `file malicious_sample.exe`
    - **Output** might look like:
    - `malicious_sample.exe: PE32 executable (GUI) Intel 80386, for MS Windows`

      This tells us the file is a **PE32 (Portable Executable)** file, which is a common format for Windows executables.

Step 2: Strings Analysis

Malware often contains readable strings that can provide useful information, such as URLs, file paths, registry keys, or commands.

1. **Run Strings Analysis**:
    - Use a tool like **strings** (Linux/macOS) or **BinText** (Windows) to extract printable strings from the executable.
    - **Command** on Linux/macOS:
    - `strings malicious_sample.exe > strings_output.txt`

    - On Windows, you can use **Strings** from **Sysinternals Suite** to get the same output:
    - `strings malicious_sample.exe > strings_output.txt`
2. **Examine Extracted Strings**:
    - Open `strings_output.txt` and examine the extracted strings. You might find interesting clues like:
        - **URLs** or **IP addresses** (common in malware command and control)
        - **File paths** (e.g., `C:\Windows\Temp\malicious.exe`)
        - **Registry keys** (e.g., `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`)
        - **Error messages or debug information** that can suggest the malware's behavior or origin.

Step 3: File Header Analysis (PE Header for Executables)

If the file is a Windows executable (PE file), the next step is to examine its **PE (Portable Executable) header**. The header contains important metadata about the executable, such as its entry point, libraries used, and more.

1. **Using PEiD for PE Header Analysis**:
    - Download **PEiD** (a popular tool for identifying packers used to compress or obfuscate files).
    - Open the suspicious file with PEiD and analyze the following:
        - **File packing**: PEiD can help identify if the file is packed using a common packer (e.g., UPX, ASPack, etc.).
        - **Compiler used**: PEiD can also indicate which compiler was used to generate the executable.
    - If a **packer** is detected, it's an indication that the file might be obfuscated to avoid detection. You'll need to unpack the file before analyzing it further.
2. **Examine the File's Imports and Exports**:
    - You can use tools like **Dependency Walker** (for Windows) to examine the **import table** of the PE file.
        - The **import table** lists all the libraries and functions that the executable calls. This can provide insight into what system resources or APIs the file might use, such as:
            - `CreateFile` (for file manipulation)
            - `WinINet` or `InternetOpenUrl` (for network connections)

Step 4: Hexadecimal Analysis

In addition to examining strings and headers, the file can also be examined in **hex** to look for embedded patterns, such as hardcoded IP addresses, suspicious hex patterns, or even embedded resources.

1. **Open File in a Hex Editor**:

- Open the suspicious file using a **Hex Editor** like **HxD** or **010 Editor**.

2. **Search for Patterns**:
   - Look for:
     - **Hardcoded IP addresses** or **domain names** (e.g., `192.168.1.100`, `www.evil.com`).
     - **Suspicious byte patterns**: Some malware can be recognized by specific byte patterns or signatures.
     - **Embedded URLs**, **file paths**, or **strings**.
3. **Check for Packed Sections**:
   - If you see large sections of **non-readable** data or repetitive patterns, it might indicate that the file is packed or encrypted.

Step 5: File Signature Analysis

Malware authors often alter file signatures or obfuscate them to avoid detection by traditional antivirus tools.

1. **Check File Signature**:
   - Use a tool like **TrID** or **FileSignatures** to check for the file's signature and compare it against known file types.
   - Example command using **TrID**:
   - `trid malicious_sample.exe`
   - The output will give you a **probability-based** guess of the file type. If it deviates from what you expect (e.g., a PE file is detected as something else), it could be an indicator of tampering or obfuscation.

Step 6: Antivirus and Heuristic Analysis

Static analysis can also include running the file through various antivirus engines or heuristic analysis tools to detect known malware signatures.

1. **Use VirusTotal**:
   - Upload the suspicious file to **VirusTotal** for a multi-antivirus scan.
   - Check for any flags from antivirus vendors regarding this file.
   - **Output** may show:
     - **Detected signatures**: If the file is known to be malicious, antivirus engines will flag it.
     - **Heuristic analysis**: Some antivirus engines might flag the file based on suspicious behavior or unusual patterns.

Step 7: Additional Static Analysis Using Resources

1. **YARA Rules**:
   - **YARA** is a tool used to identify and classify malware by looking for patterns or specific characteristics in the file.

   - Write custom **YARA rules** to search for known malware patterns or suspicious behavior in the file.
2. **Static Analysis Using Sandboxes (Optional)**:
   - While this is typically more dynamic analysis, some tools like **Cuckoo Sandbox** provide a static component where you can inspect the behavior of suspicious files before running them in an isolated environment.

Step 8: Documenting Findings

Once the static analysis is complete, document your findings:

1. **File Type and Format**:
   - What type of file is it? (e.g., PE executable, script, document)
2. **Strings and Embedded Data**:
   - What interesting strings or URLs did you find? (e.g., hardcoded IPs, domain names, suspicious file paths)
3. **File Packing**:
   - Is the file packed? If yes, what packing method was used?
4. **Imports and Exports**:
   - What system libraries does the file import? Does it use networking APIs, file system manipulation, etc.?
5. **Indicators of Compromise**:
   - Any clear indicators that suggest the file is malicious, such as connection attempts to known malicious IPs or embedded payloads.

Conclusion

Static analysis of a suspicious file can reveal a lot of important information before running the file in a live environment. By examining the file's format, extracting strings, analyzing the PE header, and identifying patterns in the raw hex data, you can gather significant intelligence about the file's behavior and potential malicious intent.

In this example, we used tools like `strings`, PEiD, Hex Editors, and VirusTotal to analyze a suspicious executable. With this information, you can make an informed decision about whether the file is safe, if it needs to be quarantined, or if further analysis is required.

PRACTICAL 9 B.

Aim: Performing dynamic analysis of a malicious file to find the processes It starts, network operations, file changes and other activities.

Code:

Malware Forensics: Dynamic Analysis of a Malicious File

Dynamic analysis involves executing the malicious file in a controlled environment (like a sandbox or virtual machine) to observe its behavior and understand its impact. This type of analysis helps identify processes it spawns, network activity, file system changes, registry modifications, and other suspicious behavior that may not be immediately apparent through static analysis.

The goal of dynamic analysis is to determine what actions the malware performs when it is run, such as:

- **Launching malicious processes**
- **Communicating with remote servers** (e.g., C&C servers)
- **Modifying system files or configurations**
- **Persisting across reboots**
- **Exploiting vulnerabilities or hijacking system resources**

Step-by-Step Dynamic Analysis

Let's assume you have a suspicious file, `malicious_sample.exe`, that you want to dynamically analyze. Below are the steps for performing dynamic analysis using a combination of tools to capture system activities.

Step 1: Set Up a Controlled Environment

Before running the malicious file, ensure that you have a controlled environment. The best practice is to use a **virtual machine (VM)** or a **sandbox** to prevent the malware from affecting the host system.

- **Virtual Machine**: Tools like **VMware** or **VirtualBox** allow you to create isolated environments where you can run malware safely.
- **Sandbox**: Platforms like **Cuckoo Sandbox** provide an environment to analyze files dynamically and capture system activity automatically.

For this walkthrough, we will focus on a **VM environment**, but many of the principles can be applied to a sandbox as well.

1. **Create a Virtual Machine**:
    - Set up a VM with an **isolated network** to prevent malware from contacting external systems (if you don't want it to communicate outside the controlled environment).
    - Install a clean version of the **Windows OS** (or any target OS, depending on the malware's platform).
2. **Snapshot the VM**:

   - Take a snapshot before running the file, so you can revert back to a clean state after analysis.
3. **Disable Antivirus** (in the VM):
   - Temporarily disable antivirus software inside the VM to ensure it doesn't interfere with the analysis.

Step 2: Monitor Processes and System Activity

Use system monitoring tools to observe any **new processes** or **suspicious activities** initiated by the malware.

1. **Monitor Processes with Process Explorer**:
   - **Process Explorer** (from Sysinternals) allows you to monitor real-time process activity, including new processes spawned by the malware.
   - Start **Process Explorer** before running the file, and watch for any new processes that might appear after execution.
     - Pay attention to processes running from suspicious locations (e.g., `C:\Users\Temp\malicious.exe`).
     - Look for unusual process names or processes that seem to execute other suspicious commands.
2. **Monitor Registry Changes with Regshot**:
   - **Regshot** is a simple registry comparison tool. Take a snapshot of the registry before running the file and take another snapshot after execution. It will show all registry changes (including new keys or modifications).
   - Key things to watch for:
     - **Persistence mechanisms**: If the malware creates or modifies registry keys for autostart (e.g., under `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`).
     - **Network settings**: Look for modifications in the registry related to networking or proxy settings.
3. **Monitor Running Processes with Task Manager**:
   - In addition to Process Explorer, you can also use **Windows Task Manager** to observe CPU, memory, and network usage spikes as suspicious files run.

Step 3: Capture Network Activity

Malware often communicates with remote servers (e.g., to download additional payloads or send stolen data). To capture this behavior, we'll use network monitoring tools.

1. **Monitor Network Connections with Wireshark**:
   - **Wireshark** is a powerful tool for monitoring network traffic in real-time.
   - Start Wireshark before executing the malicious file to capture all network activity. Filter traffic by protocols (e.g., HTTP, DNS, or ICMP) to focus on potentially malicious activity.
   - Look for:

        - **Outbound connections** to suspicious IP addresses or domains (possibly command and control servers).
        - **Unexpected DNS queries** for domains that seem unusual or are related to known malicious sources.
        - **Data exfiltration** patterns, such as large volumes of outbound traffic after the file is executed.
2. **Using TCPView**:
    - **TCPView** (from Sysinternals) is a lightweight tool that shows active network connections, including open ports and IP addresses.
    - Run **TCPView** alongside Wireshark to get a high-level view of network connections made by the suspicious file.
3. **Monitor with Netstat**:
    - Run **Netstat** (Network Statistics) to display network connections and routing tables.
    - Example command:
    - `netstat -anob`
    - This shows all active connections along with the associated processes, which can help you identify connections initiated by the malware.

Step 4: File System Monitoring

Malware often modifies or creates files to maintain persistence or store stolen data. Monitoring file system activity helps to detect these changes.

1. **Monitor File System Changes with Sysinternals Suite**:
    - Use **Process Monitor** (Procmon) from Sysinternals to capture real-time file system and registry activity.
    - Configure **Procmon** to track file system writes, creations, and deletions. Filter for suspicious behavior like the creation of new files in `C:\Windows\Temp` or modifications to executable files.
    - Look for:
        - **File drops**: Malware may create a copy of itself in a system directory or the `AppData` folder.
        - **File modifications**: Malware may modify legitimate files, especially system files or security tools.
2. **Monitor File Activity with PowerShell**:
    - Use PowerShell to track file changes, especially if you are looking for specific files or directories:
    - `Get-FileHash "C:\Path\To\Directory\*"`
    - This allows you to compare file hashes before and after execution to detect any changes.

Step 5: Analyze Behavior Using Sandboxing Tools (Optional)

If you have access to a **sandboxing environment** like **Cuckoo Sandbox**, you can automate many of these dynamic analysis steps. Cuckoo performs the following:

- **Behavioral analysis**: Tracks processes, network traffic, file system activity, and API calls.
- **Reports**: Generates detailed reports that summarize all the activities observed during the execution.

Step 6: Analyze Behavior Based on Findings

Once the malware runs in the controlled environment, review the data you've collected:

1. **Process Monitoring**:
   - Identify any unusual or new processes. These could be malware processes attempting to hide or communicate with external servers.
2. **Registry Modifications**:
   - Look for persistence mechanisms in the registry. For example, if you find entries like:
   - `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`

     This means the malware is trying to ensure it runs every time the system boots up.
3. **Network Connections**:
   - Review the network logs to identify any outbound connections to suspicious IPs. These could be attempts to exfiltrate data or communicate with a command-and-control server.
4. **File Changes**:
   - If the malware created new files, deleted files, or modified system files, this could indicate it is trying to install itself or make other changes to the system.

Step 7: Post-Analysis Activities

After completing the analysis:

- **Revert the VM to the clean snapshot** you took earlier to remove the malware from the environment.
- **Document your findings**: Report the processes, network connections, file changes, and registry modifications observed during the analysis.
- **Submit the malware** to malware intelligence repositories (e.g., VirusTotal) for further analysis and community detection.

INVESTIGATING EMAIL CRIMES

PRACTICAL 10 A.

Aim: Recovering Deleted Emails Using the Recover My Email utility.

Code:

Recovering Deleted Emails Using the Recover My Email Utility

Email deletion, whether accidental or intentional, can lead to the loss of important communications. Fortunately, in many cases, deleted emails can still be recovered using various email recovery tools, such as **Recover My Email**.

Recover My Email is a user-friendly utility that helps recover deleted or lost emails from popular email clients like Outlook, Thunderbird, and others. It scans your system for remnants of deleted emails and allows you to restore them, even if they've been emptied from the Trash or Deleted Items folder.

Step-by-Step Process to Recover Deleted Emails Using Recover My Email

Step 1: Download and Install Recover My Email Utility

1. **Download the Tool**:
    - Visit the official website for **Recover My Email** and download the utility.
    - Ensure you download the version that is compatible with your system (Windows or Mac).
2. **Install the Software**:
    - Run the installation file and follow the on-screen instructions.
    - Once installed, launch the tool.

Step 2: Select Email Program for Recovery

When you launch **Recover My Email**, the tool will prompt you to choose the email client from which you want to recover deleted emails.

1. **Select the Email Client**:
    - **Microsoft Outlook**: If you use Outlook, select it from the available options.
    - **Thunderbird**: If you use Mozilla Thunderbird, select it.
    - **Windows Mail**: If you use Windows Mail, select it.
    - **Others**: The tool also supports many other email clients, so choose the one where you want to recover emails.
2. **Choose the Email Account**:
    - If you use multiple email accounts in your email client, select the one you want to recover emails from.

Step 3: Choose the Recovery Method

The next screen will ask you how you want to recover deleted emails. **Recover My Email** typically offers a couple of recovery options:

1. **Scan the Entire Mailbox**:

   - This option scans the entire mailbox, including any folders where the deleted emails might have resided, such as the **Inbox**, **Sent**, **Trash**, **Deleted Items**, and **Spam** folders.
2. **Scan the Deleted Folder**:
   - If you know the emails were in the **Deleted Items** or **Trash** folder, you can select this option to focus on that folder specifically.
3. **Advanced Recovery** (if available):
   - This feature allows you to perform a more in-depth recovery, scanning the email client's storage area for any remnants of deleted emails. It's useful if emails have been permanently deleted or emptied from the Trash.

Step 4: Start the Scanning Process

1. **Initiate Scan**:
   - Once you've chosen the email client and recovery method, click the **Scan** button.
   - **Recover My Email** will begin scanning your system for deleted emails. This process can take a few minutes depending on the size of the email account and the volume of emails.
2. **Wait for the Scan to Complete**:
   - The tool will display a progress bar or status indicator. It will also show the number of emails found during the scan.

Step 5: Review the Recovered Emails

Once the scan is completed, the tool will present a list of the recovered emails.

1. **Preview Recovered Emails**:
   - You will be able to preview the recovered emails, including their **subject**, **sender**, **recipient**, and **date**.
   - If the emails are intact, the message body will be displayed. This is where you can confirm the emails you're looking for.
2. **Filter Emails** (if applicable):
   - Some tools offer filtering options that allow you to filter recovered emails by **date**, **subject**, or **sender**. This helps in narrowing down the emails you're looking for.

Step 6: Restore the Emails

Once you've located the deleted emails you wish to recover, you can restore them.

1. **Select Emails to Restore**:
   - Check the box next to the emails you want to recover. You can select all emails or choose specific ones.
2. **Choose Recovery Location**:
   - The tool may ask you where you want to restore the emails. You can either restore them directly to your email client or to a folder on your computer.

- **Restore to Email Client**: This option will attempt to restore the deleted emails directly into the selected email client (e.g., Outlook or Thunderbird).
- **Restore to File**: This option allows you to save the emails in a readable format like **EML**, **MSG**, or **HTML**, which can be opened in a text editor or another email client.

3. **Click Restore**:
    - Once you've made your selection, click **Restore** to recover the emails.

Step 7: Confirm Restoration

After the restoration process is complete:

1. **Check Your Email Client**:
    - Open your email client (e.g., Outlook, Thunderbird) and verify that the recovered emails are now in the desired folder (e.g., Inbox, Sent Items, etc.).
2. **Test the Recovered Emails**:
    - Open a few of the restored emails to ensure they are intact and readable.
3. **Backup**:
    - Once you've successfully recovered and verified the emails, consider backing them up to avoid future loss.

Troubleshooting Tips

- **Emails Not Found?**
    - If the tool didn't find any deleted emails, they may have been overwritten or corrupted. Consider using advanced recovery methods or running a full system scan.
- **Partial Email Recovery?**
    - If only part of the email (e.g., headers or body) is recovered, it's possible that parts of the email were corrupted during deletion or after a long period of time.
- **Tool Compatibility Issues**:
    - If the utility is not detecting your email client, ensure that your email client is correctly configured or try restarting the system and tool.

PRACTICAL 10 B.

Aim: Tracing an Email Using the eMailTrackerPro Tool.

Code:

Tracing an Email Using the eMailTrackerPro Tool

eMailTrackerPro is a powerful email tracking and tracing tool that allows you to trace the origins of an email, track the sender's location, and uncover the IP address, among other details. It's particularly useful in cyber forensics, helping investigators identify malicious emails, phishing attempts, or track down the sender's real-world location.

The process of tracing an email involves extracting information from the email headers, including the email's route and its source IP. **eMailTrackerPro** automates much of this process, making it easier to trace the path of an email message from its sender to its recipient.

Step-by-Step Guide to Tracing an Email Using eMailTrackerPro

Step 1: Download and Install eMailTrackerPro

1. **Download eMailTrackerPro**:
    - Go to the official website of **eMailTrackerPro** and download the tool. Make sure to get the version that's compatible with your operating system (Windows).
    - Download eMailTrackerPro
2. **Install eMailTrackerPro**:
    - Run the installer after downloading.
    - Follow the on-screen instructions to complete the installation.

Step 2: Obtain the Email Header

To trace an email, you need to access its **email header**, which contains critical information like the sender's IP address, routing details, and the email's path through the mail servers.

1. **Open the Email**:
    - Open the email you wish to trace in your email client (Outlook, Gmail, Yahoo, etc.).
2. **View the Email Header**:
    - **Gmail**: Click the three dots (more options) in the top-right corner of the email, then click **Show original**.
    - **Outlook**: Right-click the email, choose **Properties**, and look under the **Internet headers** section.
    - **Yahoo Mail**: Click on the three dots, select **View Full Header**.
    - **Thunderbird**: Open the email, click **More** (three horizontal lines) and choose **View Source**.
3. **Copy the Email Header**:
    - Once you've opened the full email header, select and copy the entire contents of the header.

Step 3: Open eMailTrackerPro

1. **Launch eMailTrackerPro**:
    - Open the tool after installation. You should be greeted with an interface where you can paste the email header for analysis.
2. **Paste the Email Header**:
    - In eMailTrackerPro, locate the input field for the email header. Paste the copied email header into this field.

Step 4: Trace the Email

1. **Click on "Trace"**:
    - After pasting the header, click the **Trace** button to begin analyzing the email's origin.
2. **Wait for the Analysis to Complete**:
    - eMailTrackerPro will now process the email header, extracting the sender's IP address, mail servers involved in routing the email, geographical information, and more. This may take a few moments.

Step 5: Review the Results

Once the trace is complete, eMailTrackerPro will display detailed information, including:

1. **Sender's IP Address**:
    - The tool will extract the **IP address** from the email's path, showing where the email originated.
    - If the email passed through multiple servers, the tool will list each server and its IP.
2. **Geolocation of the Sender**:
    - eMailTrackerPro can provide the **geographical location** of the IP address that sent the email. This includes the country, city, and sometimes even the specific area (latitude and longitude).
3. **Routing Information**:
    - You'll see the **email's route** as it was sent from the origin to the destination. This shows the servers the email passed through, with timestamps for each step.
4. **Mail Server Details**:
    - The tool will list the **email servers** involved in routing the message (e.g., SMTP servers), including information about their location and whether they are legitimate or suspicious.
5. **Red Flags and Suspicious Activity**:
    - If the email originated from a suspicious IP address (such as a known proxy, VPN, or data center), the tool will flag it. Additionally, it can highlight if the email is originating from an unusual location for the claimed sender.

Step 6: Use Results for Further Investigation

Now that you have traced the email's origin, the information obtained can be used for further forensic analysis or legal purposes. Here's how you can use the results:

1. **IP Address Lookup**:
    - If you want to dig deeper, you can perform a **WHOIS lookup** of the sender's IP address to find the organization or service provider associated with it. This can help identify if the sender is using a fake identity or masking their real location using a VPN.
2. **Geolocation Information**:
    - The geographic data can provide insights into the sender's real-world location, such as the city, country, and possibly even the specific area or provider they used for sending the email.
3. **Review Routing for Authenticity**:
    - Analyzing the routing can reveal whether the email took an unusual or unexpected path. For example, if the email was supposed to be from a local source but went through an international server, it might suggest suspicious activity.
4. **Follow Up with Law Enforcement**:
    - If the email is part of a crime or a phishing attack, this tracing information can be handed over to **law enforcement agencies** or your organization's security team to take further action.

Step 7: Protect Against Future Phishing Attacks

While tracing emails is useful for identifying the source of malicious emails, it's also essential to apply best practices for securing your email accounts to prevent future threats.

1. **Enable Two-Factor Authentication (2FA)**:
    - Enable **2FA** on your email accounts to add an extra layer of security. This will help prevent unauthorized access, even if your login credentials are compromised.
2. **Educate Users About Phishing**:
    - Train users to recognize phishing emails. Common signs of phishing include unsolicited requests for sensitive information, poor grammar, suspicious attachments, or links that don't match the legitimate sender's URL.
3. **Use Spam Filters**:
    - Ensure that spam filters are enabled and properly configured in your email client. Many email providers offer filters that can automatically detect and block phishing emails or emails from known suspicious sources.
4. **Use a VPN**:
    - Using a **VPN** can help protect your own email communications and mask your IP address, preventing external actors from easily tracing your emails back to you.

Troubleshooting Tips

- **Unable to Trace?**
    - If the tool doesn't return any results or has trouble tracing the email, make sure the email header is complete and correctly copied. Some email clients might hide part of the header information, so make sure to extract the **full header**.
- **Invalid IP Address?**

    - If eMailTrackerPro returns an invalid or untraceable IP, it could be that the email sender is using a **VPN** or **proxy server** to hide their true location.
- **Multiple Servers Found**:
    - If the email passed through multiple servers, follow the entire route in eMailTrackerPro to identify where the email may have been modified or manipulated.

Conclusion

Using **eMailTrackerPro** is an effective way to trace an email's origin and gather forensic evidence related to the email's sender. By analyzing the email header, extracting the sender's IP address, and mapping the email's route, you can determine the geographic location of the sender, their email provider, and any suspicious activity involved in the email's delivery.

This process is invaluable in identifying phishing attacks, fraudulent emails, or other malicious email-based activities. If you're conducting a cyber investigation, tracing emails using eMailTrackerPro can provide valuable insights into the identity and location of the malicious actor.

MOBILE FORENSICS

PRACTICAL 11.

Aim: Analyzing the Forensic Image and Carving the Deleted Files Using Autopsy.

Code:

Mobile Forensics: Analyzing the Forensic Image and Carving the Deleted Files Using Autopsy

Mobile forensics involves the recovery, analysis, and preservation of digital evidence from mobile devices such as smartphones and tablets. Autopsy is a powerful, open-source forensic tool commonly used for disk forensics and image analysis. It allows investigators to analyze forensic images and recover deleted files from these images, including from mobile devices.

Pre-Requisites

1. **Autopsy Installation**: Ensure you have **Autopsy** installed on your system. You can download it from here.
2. **Forensic Image**: You should have a forensic image (such as `.dd` or `.img` file) of the mobile device or memory card you're investigating.
3. **Access to the Necessary Mobile Device Image**: You must have a **forensic image** created using an appropriate tool (like **FTK Imager**, **dd**, **Cellebrite**, or **XRY**).

Step-by-Step Process to Analyze the Forensic Image and Carve Deleted Files Using Autopsy

Step 1: Install Autopsy

1. **Download and Install Autopsy**:
   - Download the latest version of Autopsy from the official site: Autopsy Download.
   - Follow the installation instructions for your operating system (Autopsy works on Windows, Mac, and Linux).

   After installation, launch the Autopsy application.

Step 2: Create a New Case

1. **Open Autopsy**:
   - Open the Autopsy tool.
2. **Create a New Case**:
   - Click on **"Create New Case"**.
   - Enter the **Case Name** and **Case Number**. You can also specify the **Investigator** and **Case Description**.
   - Choose a directory where the case data will be stored.

Step 3: Add the Forensic Image

1. **Add Data Source**:
    - Once the case is created, click on **"Add Data Source"** to add the forensic image for analysis.
    - Select **Disk Image or VM file** since we are analyzing a forensic image.
2. **Browse to the Forensic Image**:
    - Browse to the location where you have stored the mobile device forensic image (e.g., `phone_image.dd`).
    - Select the image file, then click **Next**.
3. **Select Image Type**:
    - Autopsy will prompt you to choose the **type of image** (e.g., **Raw image**, **E01 image**, etc.). Choose the appropriate type and click **Next**.
4. **Start Analysis**:
    - After adding the image, Autopsy will begin analyzing the disk image.
    - You will be asked to select various modules to run for analysis, such as **File Analysis**, **Keyword Search**, **Hash Analysis**, and **Carving**.

Step 4: File Analysis and Initial Inspection

1. **Browse Files**:
    - After Autopsy has indexed the image, you will be able to browse through the files found within the forensic image.
    - The **File Analysis** module will display directories, files, and file metadata found in the image.
2. **Inspect Mobile Data**:
    - In the file explorer window, navigate through the folders. For mobile devices, you may see directories such as **SMS**, **Contacts**, **Call Logs**, **Photos**, etc.
3. **Review Known File Types**:
    - Autopsy automatically categorizes files into known types, such as images, documents, and messages.
    - You can also filter the results based on file types like **SMS**, **Call Logs**, or **Applications** to focus on specific artifacts.

Step 5: Carving Deleted Files

Autopsy can help recover deleted files from the image using a process called **data carving**. This process involves scanning the raw data on the image for file signatures and attempting to reconstruct deleted files.

1. **Enable Data Carving**:
    - To carve deleted files, go to the **"Module"** section in Autopsy.
    - Select **"Data Carving"**. Data carving looks for file signatures (like `.jpg`, `.mp3`, `.apk`, etc.) in unallocated space, which is where deleted files might still reside.
2. **Configure Carving Settings**:
    - You can specify which file types to carve based on extensions or file signatures.
    - For example, if you are looking for images, you can select the **JPEG** file type, or for mobile app data, you can choose **APK** or **SQLite**.

3. **Start Carving**:
    - Click **Start Carving** to begin the process of recovering deleted files.
    - Autopsy will scan the unallocated space of the image for remnants of deleted files. This can take some time depending on the size of the image.
4. **View Carved Files**:
    - Once the carving is completed, Autopsy will display any recovered files in the **"Carved Files"** section.
    - These files may include **deleted images**, **documents**, **application data**, and more.

Step 6: Analyze Carved Files

1. **Inspect the Recovered Files**:
    - Once the carving process is complete, you can browse through the **Carved Files** section in Autopsy.
    - You may find files that were deleted but still partially or fully recoverable from the disk image.
2. **Open and Verify the Files**:
    - You can double-click on the carved files to view their content. Autopsy will open them in a suitable viewer (e.g., image viewer, text editor, etc.).
3. **Export the Files**:
    - If you want to export the carved files, right-click on the file and select **Export**. Choose a location to save the recovered files.

Step 7: Report Findings

1. **Generate a Report**:
    - Once you've completed the analysis and recovered the necessary data, you can generate a forensic report.
    - Go to **"Reports"** in Autopsy and choose the format (e.g., HTML, CSV, or PDF) for your report.
    - Include details about the carved files, any identified artifacts, and analysis results.
2. **Save and Share the Report**:
    - The report will contain all the evidence, including recovered deleted files, metadata, and any relevant findings from the image analysis.
    - Save and share the report with relevant parties, such as law enforcement or legal teams.

Code Example for Using Autopsy (CLI)

While Autopsy is primarily a graphical interface tool, some operations (like **carving files**) can be automated with scripts. Autopsy is built on **The Sleuth Kit (TSK)**, and you can use TSK tools for advanced scripting.

Here is an example of a basic **data carving script** using **TSK** tools that can be run in conjunction with Autopsy:

1. **Run TSK's `fls` Command** to list files and directories:

```
fls -r /path/to/image.dd > file_list.txt
```

2. **Run TSK's `icat` Command** to extract a specific file by inode:

```
icat /path/to/image.dd <inode_number> > extracted_file
```

3. **Run `tsk_recover`** to carve deleted files:

```
tsk_recover /path/to/image.dd /path/to/output_directory
```

These commands can be used to supplement the carving process within Autopsy and extract data from raw disk images.

Conclusion

Using **Autopsy** for mobile forensics is an excellent method for analyzing forensic images, recovering deleted files, and carving data from a mobile device's disk image. The process involves creating a case, adding the forensic image, running file analysis, and using data carving techniques to recover deleted files.

Autopsy, along with **TSK tools**, provides a comprehensive suite for both **image analysis** and **file recovery**, making it a powerful tool for investigators working on mobile forensics cases.