PRACTICAL JOURNAL

in

BLOCKCHAIN

Submitted to

Laxman Devram Sonawane College, Kalyan (W) 421301

in partial fulfilment for the award of the degree of

Master of Science in Information Technology

(Affiliated to Mumbai University)

Submitted by

Rithik Panneerselvam Nadar

Under the guidance of

Ms. Sabina Ansari

Department of Information Technology
Kalyan, Maharashtra

Academic Year 2024-25

The Kalyan Wholesale Merchants Education Society's

Laxman Devram Sonawane College,

Kalyan (W) 421301

Department of Information Technology
Masters of Science – Part II

Certificate

This is to certify that **Mr. Rithik Panneerselvam Nadar**, Seat number_______, studying in Masters of Science in Information Technology Part II, Semester IV has satisfactorily completed the practical of "**BLOCKCHAIN**" as prescribed by University of Mumbai, during the academic year 2024-25.

Subject In-charge	Coordinator In-charge	ExternalExaminer

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.	
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	
4	A	Install and demonstrate use of hyperledger-Irhoa.	

Practical 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.

```
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
import binascii
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer = PKCS1_v1_5.new(self._private_key)

self.identity=binascii.hexlify(self._public_key.exportKey(format='DER')).decode('asc
ii')

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

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

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

client1 = Client()
client2 = Client()

print (client1.identity)
print("*******")
print (client2.identity)
```

Python 3.9.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/WORK/MSC IT/blockchain/new1a.py ==================
30819f300d06092a864886f70d010101050003818d0030818902818100ca4df6f1359007ca67d310
d651571873216de776f5a44babc7034b6221bc0703b297918ef31f1fecfec1180c1b959eb11f432a
12ce9230b66668f981d3b0605e83d9baa90938d95ddf9c50a4c42b6fc36aaf462f0c3c713b6680b8
d78eea3398339dd55011953bb640ab0faa58692b0efc080a989d10117303db652600edb555020301
0001
*******
30819f300d06092a864886f70d010101050003818d0030818902818100bd3177c05ad3b6bd60c0cd
e592fcf633b7332666e1533712a3274f3f751110b9c40404af33a721c2021742b2a7a6d4202929db
f6e7e0d8cab39b9fb0c7b83cd12542fb58aee3676b12024687339a87bbaffacae454cd17c7b17e6a
94a5e7afccd3d279fc400e32e8a9f28075cd2e040f0d9c39d65cff47439a869c6572ad9975020301
0001
>>>
```

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..

```
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
import binascii
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer = PKCS1_v1_5.new(self._private_key)

self.identity=binascii.hexlify(self._public_key.exportKey(format='DER')).decode('asc
ii')

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

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

```
            'time': self.time})

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

Manisha = Client()
Karan = Client()
t = Transaction (sender=Manisha,recipient=Karan.identity,value=5.0)
signature = t.sign_transaction()
print (signature)

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

transactions = []
Manisha = Client()
Karan = Client()
Seema = Client()
Priti = Client()
t1 = Transaction( Manisha, Karan.identity, 15.0 )
t1.sign_transaction()
transactions.append(t1)
t2 = Transaction( Manisha, Seema.identity, 6.0 )
t2.sign_transaction()
transactions.append(t2)
t3 = Transaction( Karan, Priti.identity, 2.0)
t3.sign_transaction()
transactions.append(t3)
t4 = Transaction( Seema, Karan.identity, 4.0 )
t4.sign_transaction()
transactions.append(t4)
t5 = Transaction( Priti, Seema.identity, 7.0 )
t5.sign_transaction()
transactions.append(t5)
t6 = Transaction( Karan, Seema.identity, 3.0 )
t6.sign_transaction()
transactions.append(t6)
t7 = Transaction( Seema, Manisha.identity, 8.0 )
```

```
t7.sign_transaction()
transactions.append(t7)
t8 = Transaction( Seema, Karan.identity, 1.0 )
t8.sign_transaction()
transactions.append(t8)
t9 = Transaction( Priti, Manisha.identity, 5.0 )
t9.sign_transaction()
transactions.append(t9)
t10 = Transaction( Priti, Karan.identity, 3.0 )
t10.sign_transaction()
transactions.append(t10)

for transaction in transactions:
   display_transaction (transaction)
print ('=============================')
```

```
=====================
value:3.0
=====================
time:2025-06-06 10:34:55.269748
=====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100c50fb314c9152a27cf1b6dad29733abba4da38490e3c265b8e319015b4ca9a536beda027856e77a585de066f579a6bace1a574da151c1c4b2fb4bbdf79d
3e0fd303308e65d8101925b0bde02e5f327b1f4023013a98855ce1d9df9306797e7a032b009618b33796d68ec76e5dea839d5ffccf29bbc8b28cd27b5cbdb72fbf1110203010001
=====================
recipient:30819f300d06092a864886f70d010101050003818d0030818902818100b145f2ca4ac8554f91c8892b3a3b738256bf14e5ef47cfdd3b1d65f224063c9d1672beba8c5cec5d29e8906d36fea2afaa7fc331d4287f5c4947bbef
db10060ae7f6cc3d40965eb94d2f9c43118aa44f1cee84e22016980e4bd56aee297b2aa3aef729b1d846099c427c2b9d2625c642d8e98bd1663fdac66381d2206c2c4a2f0203010001
=====================
value:8.0
=====================
time:2025-06-06 10:34:55.269748
=====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100c50fb314c9152a27cf1b6dad29733abba4da38490e3c265b8e319015b4ca9a536beda027856e77a585de066f579a6bace1a574da151c1c4b2fb4bbdf79d
3e0fd303308e65d8101925b0bde02e5f327b1f4023013a98855ce1d9df9306797e7a032b009618b33796d68ec76e5dea839d5ffccf29bbc8b28cd27b5cbdb72fbf1110203010001
=====================
recipient:30819f300d06092a864886f70d010101050003818d0030818902818100a806fcd24e88a73860feb3e9a06cafb8edf70f6a1f3f38b3073e697cf303baa6f821ca4bb731368c1b3c0bbf90e53916bea46343eacc0b16721ae333
5bbf971e56f482257caaf4c3df83a0ada947d2b136ab2f4758385a46ab83aa0781115e76be66f96b4f7392fedf2be95dbeec0f37a6e33b126031563367cbec5087694ba10203010001
=====================
value:1.0
=====================
time:2025-06-06 10:34:55.269748
=====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100d3b8f4a714a2b3bfd810342eafb871ca6c3341eff6b52873a4287a56cc8058b6bc75456866adfd87480719a976498843cc50df3508c8069eacb92348d4a
2004a4f0ffe3213e0a34e9e65c3caaef1ccbf11fe885041e5643939845cee2b14a09a3789b56407df1281bc24273f09bed98ec69b0285860f5177522c31d30f37f3470203010001
=====================
recipient:30819f300d06092a864886f70d010101050003818d0030818902818100b145f2ca4ac8554f91c8892b3a3b738256bf14e5ef47cfdd3b1d65f224063c9d1672beba8c5cec5d29e8906d36fea2afaa7fc331d4287f5c4947bbef
db10060ae7f6cc3d40965eb94d2f9c43118aa44f1cee84e22016980e4bd56aee297b2aa3aef729b1d846099c427c2b9d2625c642d8e98bd1663fdac66381d2206c2c4a2f0203010001
=====================
value:5.0
=====================
time:2025-06-06 10:34:55.269748
=====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100d3b8f4a714a2b3bfd810342eafb871ca6c3341eff6b52873a4287a56cc8058b6bc75456866adfd87480719a976498843cc50df3508c8069eacb92348d4a
2004a4f0ffe3213e0a34e9e65c3caaef1ccbf11fe885041e5643939845cee2b14a09a3789b56407df1281bc24273f09bed98ec69b0285860f5177522c31d30f37f3470203010001
=====================
recipient:30819f300d06092a864886f70d010101050003818d0030818902818100a806fcd24e88a73860feb3e9a06cafb8edf70f6a1f3f38b3073e697cf303baa6f821ca4bb731368c1b3c0bbf90e53916bea46343eacc0b16721ae333
5bbf971e56f482257caaf4c3df83a0ada947d2b136ab2f4758385a46ab83aa0781115e76be66f96b4f7392fedf2be95dbeec0f37a6e33b126031563367cbec5087694ba10203010001
=====================
value:3.0
=====================
time:2025-06-06 10:34:55.269748
=====================
```

D. Implement a function to add new blocks to the miner and dump the blockchain.

```
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
import binascii

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

self.identity=binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')

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

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

    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
```

```
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')

Manisha = Client()
Karan = Client()
t = Transaction (sender=Manisha,recipient=Karan.identity,value=5.0)
signature = t.sign_transaction()
print (signature)
def display_transaction(transaction):
    #for transaction in transactions:
    dict = transaction.to_dict()
    print ("sender:" + dict['sender'])
    print ('====================')
    print ("recipient:" + dict['recipient'])
    print ('====================')
    print ("value:" + str(dict['value']))
    print ('====================')
    print ("time:" + str(dict['time']))
    print ('====================')

transactions = []
Manisha = Client()
Karan = Client()
Seema = Client()
Priti = Client()
t1 = Transaction( Manisha, Karan.identity, 15.0 )
t1.sign_transaction()
transactions.append(t1)
t2 = Transaction( Manisha, Seema.identity, 6.0 )
t2.sign_transaction()
transactions.append(t2)
t3 = Transaction( Karan, Priti.identity, 2.0)
t3.sign_transaction()
transactions.append(t3)
t4 = Transaction( Seema, Karan.identity, 4.0 )
t4.sign_transaction()
transactions.append(t4)
t5 = Transaction( Priti, Seema.identity, 7.0 )
t5.sign_transaction()
transactions.append(t5)
t6 = Transaction( Karan, Seema.identity, 3.0 )
t6.sign_transaction()
transactions.append(t6)
t7 = Transaction( Seema, Manisha.identity, 8.0 )
t7.sign_transaction()
transactions.append(t7)
t8 = Transaction( Seema, Karan.identity, 1.0 )
t8.sign_transaction()
transactions.append(t8)
t9 = Transaction( Priti, Manisha.identity, 5.0 )
```

```
t9.sign_transaction()
transactions.append(t9)
t10 = Transaction( Priti, Karan.identity, 3.0 )
t10.sign_transaction()
transactions.append(t10)

for transaction in transactions:
    display_transaction (transaction)
print ('=================================')

class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""
last_block_hash = ""
Manisha = Client()
t0 = Transaction ( "LDS", Karan.identity, 500.0 )

block0 = Block()
block0.previous_block_hash = None
Nonce = None
block0.verified_transactions.append (t0)
digest = hash (block0)
last_block_hash = digest
TPCoins = []

def dump_blockchain (self):
    print ("Number of blocks in the chain:" + str(len (self)))
    for x in range (len(TPCoins)):
        block_temp = TPCoins[x]
        print ("block #" + str(x))
        for transaction in block_temp.verified_transactions:
            display_transaction (transaction)
            print ('==============================================')
        print ('==============================================')
TPCoins.append (block0)
dump_blockchain(TPCoins)
```

```
time:2025-06-06 10:36:39.923717
====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100e68bf736d3664542d08f40ca4ebcaa8aa5fb411d0a62f03f7c32a29d262b828f86fd136125be2ba35c32f14a78d7088eb96c473de93628f4678336cec7b
b42b2fbf497f118faedc5ccf170eb4124e054858ff8b9b7d59775a0fe24cece4dcc88baa290dd2144b083c917611c3178245ede709f48f8ab967b2da7f0b2d35b74ab0203010001
====================
recipient:30819f300d06092a864886f70d010101050003818d0030818902818100f3f58e3950bf07934149de06e06490907876fd21c9fb8228ffe5f6b5742d969b4ab38dc9b5c83bf576092a42e7cdab9a8fe99888f000626a6a6bef49
79b2b90751541cd1999c195131bb161a05093fc81a7330697d1ff576c8b8e80ffb559ba8484fc81e428d667f923a7b7a30fb5b2556074316d35cc69bf3100044af5365fb0203010001
====================
value:1.0
====================
time:2025-06-06 10:36:39.923717
====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100a93af40de2b8575c033293092ba151924538790d60fc83eb457724fff93a4bd766da27b8b17d25678ba94a2b9a057ee487a870fa0eb74f3b082038978b4
2ddb72ae9add17d718285ecb2b97e4e5fc0c260ea49e9cb61fdfdc6b5e98d3b328c93c314aab63fad19f2b37cfae3705cd084aee3d76bc0fbd66a785ed019a2cef0df0203010001
====================
recipient:30819f300d06092a864886f70d010101050003818d003081890281810096782f7c49db72935ad24527d9fe2106d1005abc8961c36cca053827cff759b495a2290fc349af5e2a54e48b3e11ab2089fa3c53881b70851b1e4a85
0185fef770e21cb95ca7372d152f612a9c67c49f5dc389dc21c4857cae5294c07c1fe7216d6ad222587b04dcd450b80e67e34538fa5b22d3994a5e3489983fc5bced95b70203010001
====================
value:5.0
====================
time:2025-06-06 10:36:39.923717
====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100a93af40de2b8575c033293092ba151924538790d60fc83eb457724fff93a4bd766da27b8b17d25678ba94a2b9a057ee487a870fa0eb74f3b082038978b4
2ddb72ae9add17d718285ecb2b97e4e5fc0c260ea49e9cb61fdfdc6b5e98d3b328c93c314aab63fad19f2b37cfae3705cd084aee3d76bc0fbd66a785ed019a2cef0df0203010001
====================
recipient:30819f300d06092a864886f70d010101050003818d0030818902818100f3f58e3950bf07934149de06e06490907876fd21c9fb8228ffe5f6b5742d969b4ab38dc9b5c83bf576092a42e7cdab9a8fe99888f000626a6a6bef49
79b2b90751541cd1999c195131bb161a05093fc81a7330697d1ff576c8b8e80ffb559ba8484fc81e428d667f923a7b7a30fb5b2556074316d35cc69bf3100044af5365fb0203010001
====================
value:3.0
====================
time:2025-06-06 10:36:39.923717
====================
==============================
Number of blocks in the chain:1
block #0
sender:LDS
====================
recipient:30819f300d06092a864886f70d010101050003818d0030818902818100f3f58e3950bf07934149de06e06490907876fd21c9fb8228ffe5f6b5742d969b4ab38dc9b5c83bf576092a42e7cdab9a8fe99888f000626a6a6bef49
79b2b90751541cd1999c195131bb161a05093fc81a7330697d1ff576c8b8e80ffb559ba8484fc81e428d667f923a7b7a30fb5b2556074316d35cc69bf3100044af5365fb0203010001
====================
value:500.0
====================
time:2025-06-06 10:36:40.788674
====================
```

Practical 2

A. Write a python program to demonstrate mining.

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

```
>>>
========================================================================= RESTART: C:/WORK/MSC IT/bloc
kchain/new2a.py =======================================================================
after 258 iterations found nonce: 11ad95a2996e2c26c523bc7b298dd0de8ee83513b167943e5d07bce55e0767f9
```

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.**

What Is A Full Node?
A full node is a program that fully validates transactions and blocks. Almost all full nodes also help the network by accepting transactions and blocks from other full nodes, validating those transactions and blocks, and then relaying them to further full nodes. Most full nodes also serve lightweight clients by allowing them to transmit their transactions to the network and by notifying them when a transaction affects their wallet. If not enough nodes perform this function, clients won't be able to connect through the peer-to-peer network—they'll have to use centralized services instead.
Many people and organizations volunteer to run full nodes using spare computing and bandwidth resources—but more volunteers are needed to allow Bitcoin to continue to grow. This document describes how you can help and what helping will cost you.

Costs And Warnings
Running a Bitcoin full node comes with certain costs and can expose you to certain risks. This section will explain those costs and risks so you can decide whether you're able to help the network.
Special Cases
Miners, businesses, and privacy-conscious users rely on particular behavior from the full nodes they use, so they will often run their own full nodes and take special safety precautions. This document does not cover those precautions—it only describes running a full node to help support the Bitcoin network in general.
Please seek out assistance in the community if you need help setting up your full node correctly to handle high-value and privacy-sensitive tasks. Do your own diligence to ensure who you get help from is ethical, reputable and qualified to assist you.
Secure Your Wallet
It's possible and safe to run a full node to support the network and use its wallet to store your bitcoins, but you must take the same precautions you would when using any Bitcoin wallet. Please see the securing your wallet page for more information.
Minimum Requirements
Bitcoin Core full nodes have certain requirements. If you try running a node on weak hardware, it may work—but you'll likely spend more time dealing with issues. If you can meet the following requirements, you'll have an easy-to-use node.

- Desktop or laptop hardware running recent versions of Windows, Mac OS X, or Linux.
- 7 gigabytes of free disk space, accessible at a minimum read/write speed of 100 MB/s.
- 2 gigabytes of memory (RAM)
- A broadband Internet connection with upload speeds of at least 400 kilobits (50 kilobytes) per second
- An unmetered connection, a connection with high upload limits, or a connection you regularly monitor to ensure it doesn't exceed its upload limits. It's common for full nodes on high-speed connections to use 200 gigabytes upload or more a month. Download usage is around 20 gigabytes a month, plus around an additional 340 gigabytes the first time you start your node.
- 6 hours a day that your full node can be left running. (You can do other things with your computer while running a full node.) More hours would be better, and best of all would be if you can run your node continuously.

Possible Problems

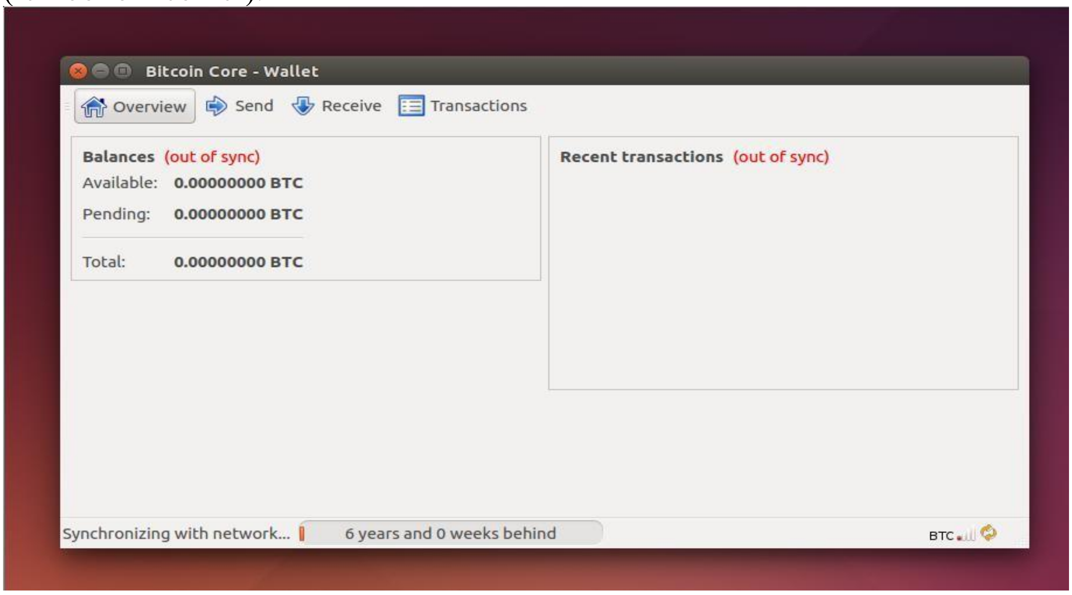
- **Legal:** Bitcoin use is prohibited or restricted in some areas.
- **Bandwidth limits**: Some Internet plans will charge an additional amount for any excess upload bandwidth used that isn't included in the plan. Worse, some providers may terminate your connection without warning because of overuse. We advise that you check whether your Internet connection is subjected to such limitations and monitor your bandwidth use so that you can stop Bitcoin Core before you reach your upload limit.
- **Anti-virus:** Several people have placed parts of known computer viruses in the Bitcoin block chain. This block chain data can't infect your computer, but some anti-virus programs quarantine the data anyway, making it more difficult to run Bitcoin Core. This problem mostly affects computers running Windows.
- **Attack target:** Bitcoin Core powers the Bitcoin peer-to-peer network, so people who want to disrupt the network may attack Bitcoin Core users in ways that will affect other things you do with your computer, such as an attack that limits your available download bandwidth.

Initial Block Download(IBD)

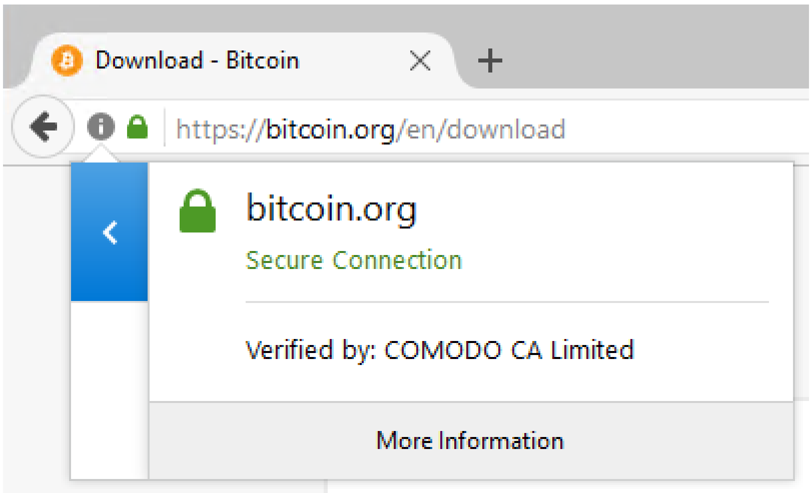
Initial block download refers to the process where nodes synchronize themselves to the network by downloading blocks that are new to them. This will happen when a node is far behind the tip of the best block chain. In the process of IBD, a node does not accept incoming transactions nor request mempool transactions.

If you are trying to set up a new node following the instructions below, you will go through the IBD process at the first run, and it may take a considerable amount of time since a new node has to download the entire block chain (which is roughly 340 gigabytes now). During the download, there could be a high usage for the network and CPU (since the node has to verify the blocks downloaded), and the client will take up an increasing amount of storage space (reduce storage provides more details on reducing storage). Before the node finishes IBD, you will not be able to see a new transaction related to your account until the client has caught up to the block containing that transaction. So your wallet may not count new payments/spendings into the balance.

If you are using Bitcoin Core GUI, you can monitor the progress of IBD in the status bar (left bottom corner).



Windows Instructions
Windows 10
Go to the Bitcoin Core download page and verify you have made a secure connection to the server.



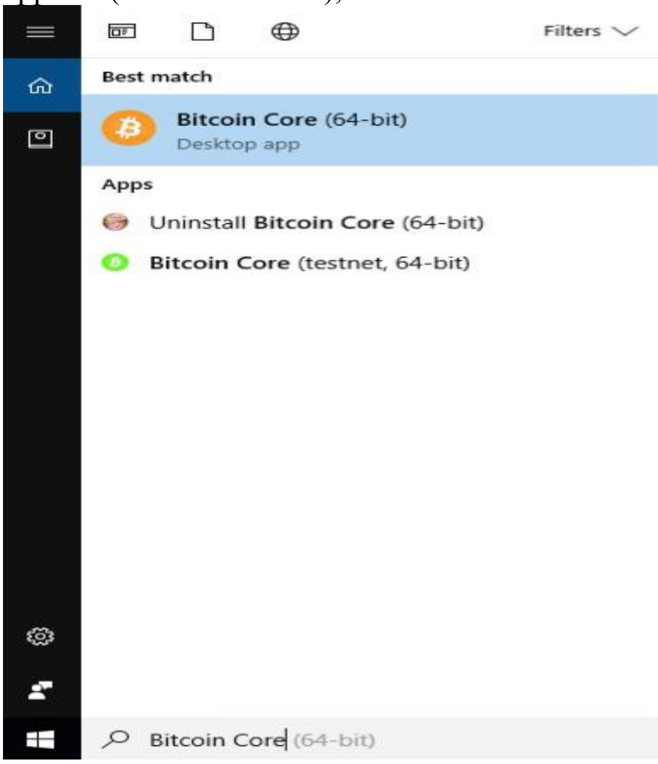
Click the large blue *Download Bitcoin Core* button to download the Bitcoin Core installer to your desktop.
If you know how to use PGP, you should also click the *Verify Release Signatures* link on the download page to download a signed list of SHA256 file hashes. The 0.11 and later releases are signed by Wladimir J. van der Laan's releases key with the fingerprint:
01EA 5486 DE18 A882 D4C2 6845 90C8 019E 36C2 E964
Earlier releases were signed by Wladimir J. van der Laan's regular key. That key's fingerprint is:
71A3 B167 3540 5025 D447 E8F2 7481 0B01 2346 C9A6
Even earlier releases were signed by Gavin Andresen's key. His primary key's fingerprint is:
2664 6D99 CBAE C9B8 1982 EF60 29D9 EE6B 1FC7 30C1
You should verify these keys belong to their owners using the web of trust or other trustworthy means. Then use PGP to verify the signature on the release signatures file. Finally, use PGP or another utility to compute the SHA256 hash of the archive you downloaded, and ensure the computed hash matches the hash listed in the verified release signatures file.
After downloading the file to your desktop or your Downloads folder (C:\Users\<YOUR USER NAME>\Downloads), run it by double-clicking its icon. Windows will ask you to confirm that you want to run it. Click Yes and the Bitcoin installer will start. It's a typical Windows installer, and it will guide you through the decisions you need to make about where to install Bitcoin Core.

To continue, choose one of the following options

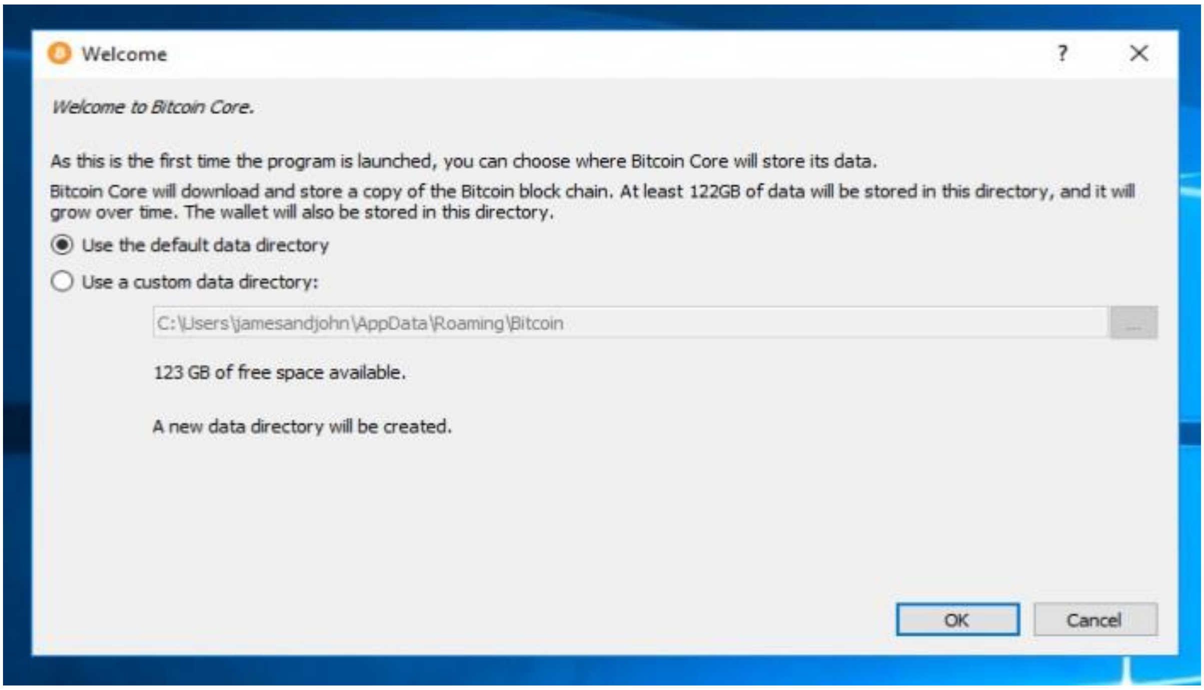
1. If you want to use the Bitcoin Core Graphical User Interface (GUI), proceed to the Bitcoin Core GUI section below.
2. If you want to use the Bitcoin Core daemon (bitcoind), which is useful for programmers and advanced users, proceed to the Bitcoin Core Daemon section below.
3. If you want to use both the GUI and the daemon, read both the GUI instructions and the daemon instructions. Note that you can't run both the GUI and the daemon at the same time using the same configuration directory.

Bitcoin Core GUI

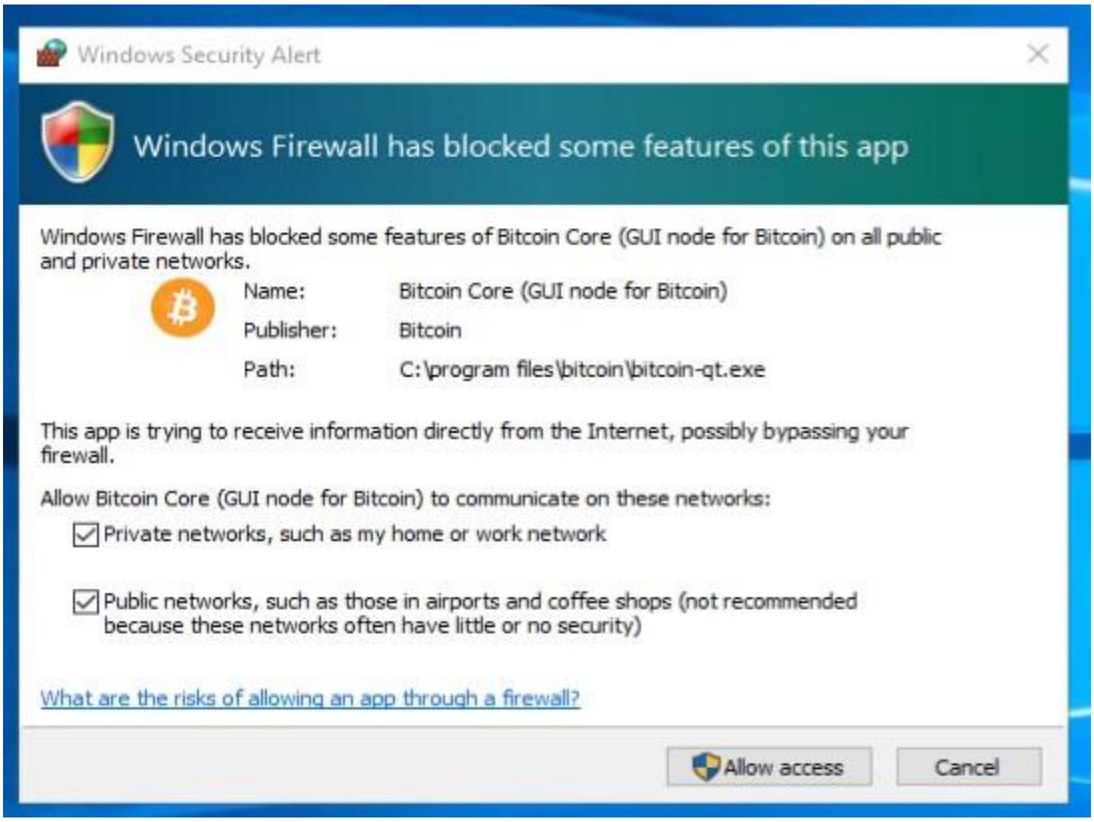
Press the Windows key (⊞ Win) and start typing "bitcoin". When the Bitcoin Core icon appears (as shown below), click on it.



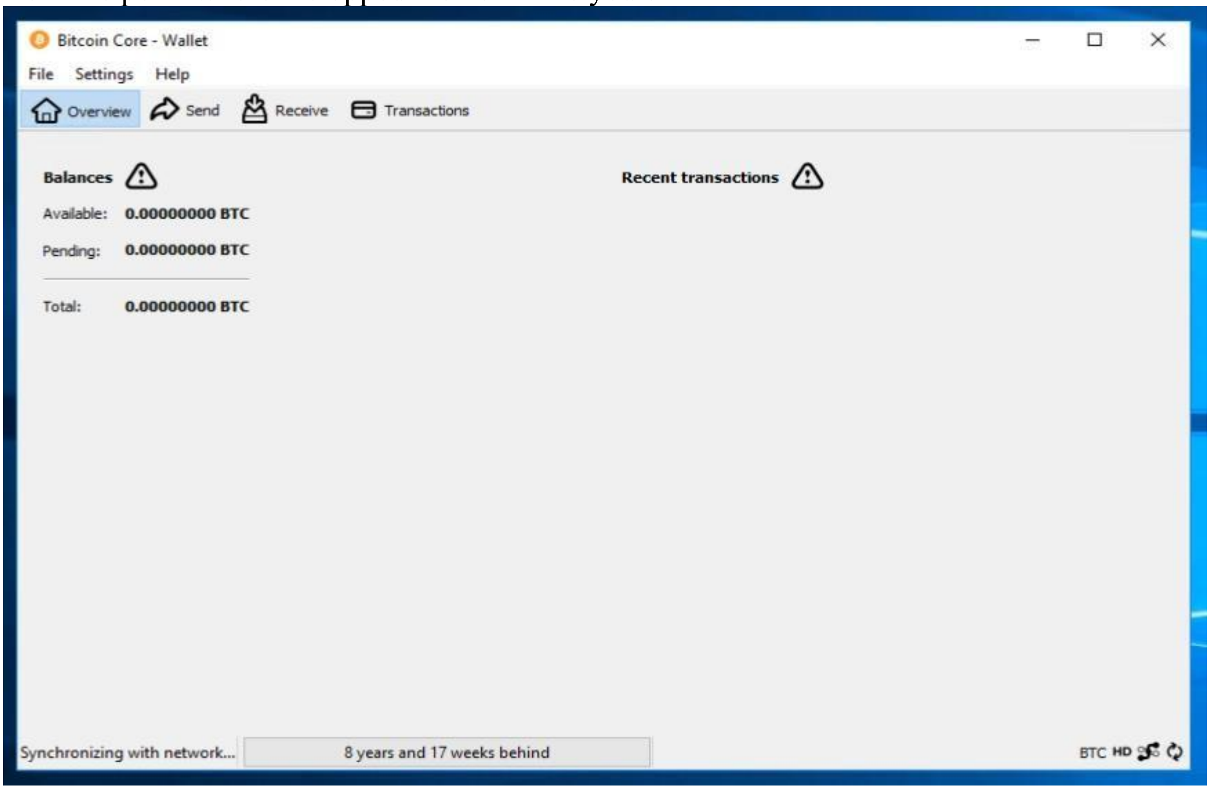
You will be prompted to choose a directory to store the Bitcoin block chain and your wallet. Unless you have a separate partition or drive you want to use, click Ok to use the default.



Your firewall may block Bitcoin Core from making outbound connections. It's safe to allow Bitcoin Core to use all networks.



Bitcoin Core GUI will begin to download the block chain. This step will take at least several days, and it may take much more time on a slow Internet connection or with a slow computer. During the download, Bitcoin Core will use a significant part of your connection bandwidth. You can stop Bitcoin Core at any time by closing it; it will resume from the point where it stopped the next time you start it.

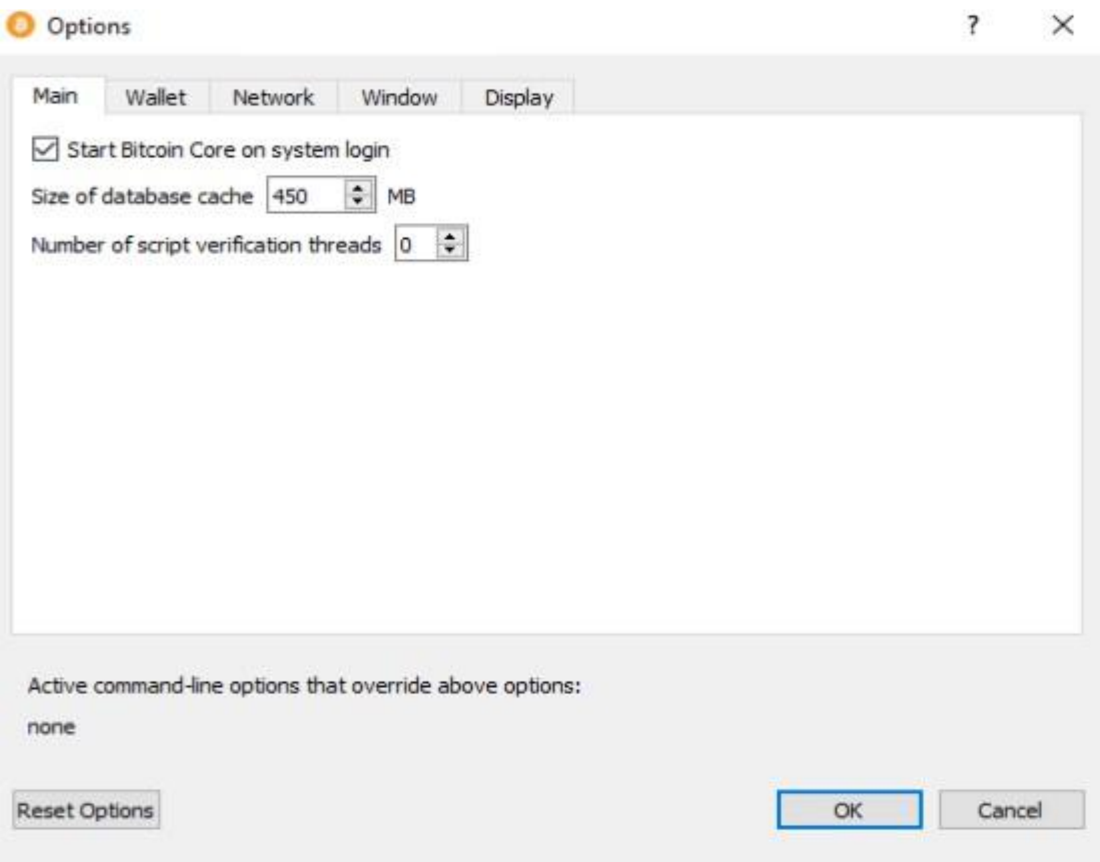


After download is complete, you may use Bitcoin Core as your wallet or you can just let it run to help support the Bitcoin network.

Optional: Start Your Node At Login

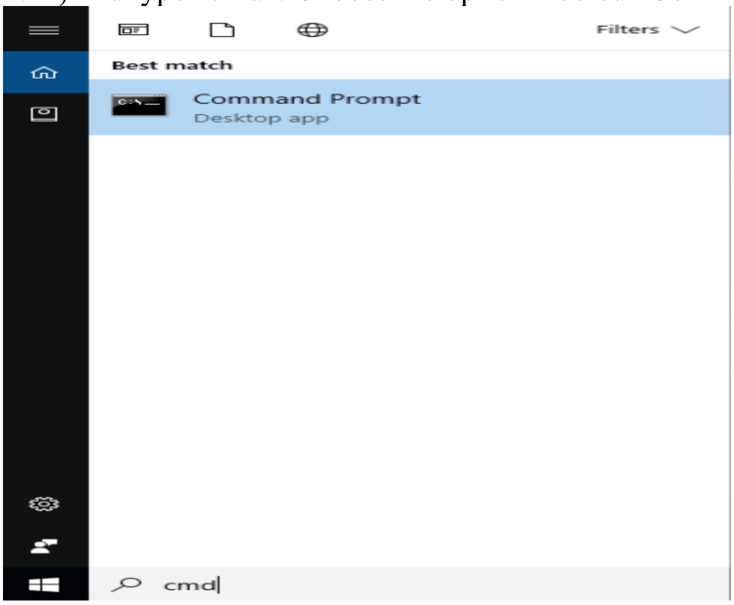
Starting your node automatically each time you login to your computer makes it easy for you to contribute to the network. The easiest way to do this is to tell Bitcoin Core GUI to start at login.

While running Bitcoin Core GUI, open the Settings menu and choose Options. On the Main tab, click *Start Bitcoin on system login*. Click the Ok button to save the new settings.



The next time you login to your desktop, Bitcoin Core GUI will be automatically started minimized in the task bar.

To start Bitcoin Core daemon, first open a command window: press the Windows key (⊞ Win) and type "cmd". Choose the option labeled "Command Prompt".



If you installed Bitcoin Core into the default directory, type the following at the command prompt:

C:\Program Files\Bitcoin\daemon\bitcoind

Bitcoin Core daemon should start. To interact with Bitcoin Core daemon, you will use the command bitcoin-cli (Bitcoin command line interface). If you installed Bitcoin Core into the default location, type the following at the command prompt to see whether it works:

C:\Program Files\Bitcoin\daemon\bitcoin-cli getblockchaininfo
Note: it may take up to several minutes for Bitcoin Core to start, during which it will display the following message whenever you use bitcoin-cli:
error: {"code":-28,"message":"Verifying blocks..."}
After it starts, you may find the following commands useful for basic interaction with your node: getblockchaininfo, getnetworkinfo, getnettotals, getwalletinfo, stop, and help.
For example, to safely stop your node, run the following command:
C:\Program Files\Bitcoin\daemon\bitcoin-cli stop
A complete list of commands is available in the Bitcoin.org developer reference.
When Bitcoin Core daemon first starts, it will begin to download the block chain. This step will take at least several days, and it may take much more time on a slow Internet connection or with a slow computer. During the download, Bitcoin Core will use a significant part of your connection bandwidth. You can stop Bitcoin Core at any time using the stop command; it will resume from the point where it stopped the next time you start it.
Optional: Start Your Node At Boot
Starting your node automatically each time your computer boots makes it easy for you to contribute to the network. The easiest way to do this is to start Bitcoin Core daemon when you login to your computer.
Start File Explorer and go to:
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp
Right-click on the File Explorer window and choose New → Text file. Name the file start_bitcoind.bat. Then right-click on it and choose Open in Notepad (or whatever editor you prefer). Copy and paste the following line into the file.
C:\Program Files\Bitcoin\daemon\bitcoind
(If you installed Bitcoin Core in a non-default directory, use that directory path instead.)
Save the file. The next time you login to your computer, Bitcoin Core daemon will be automatically started.

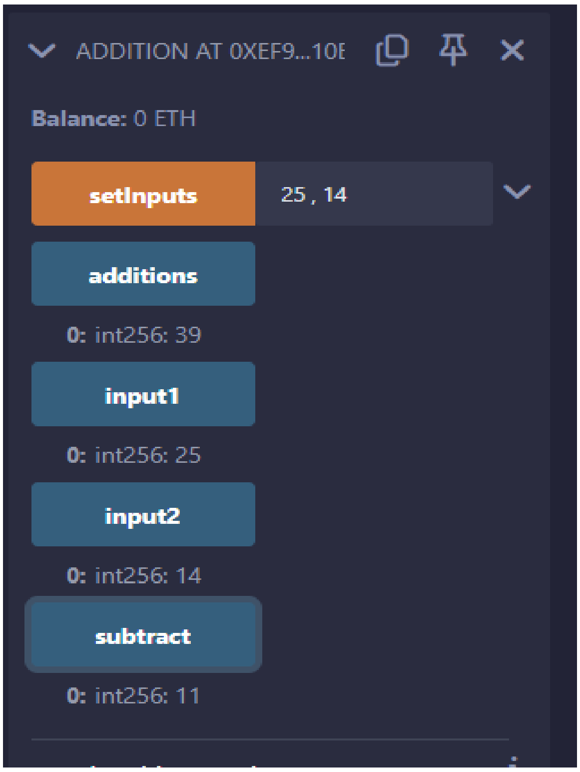
Practical 3

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

- **regular functions**

Code:

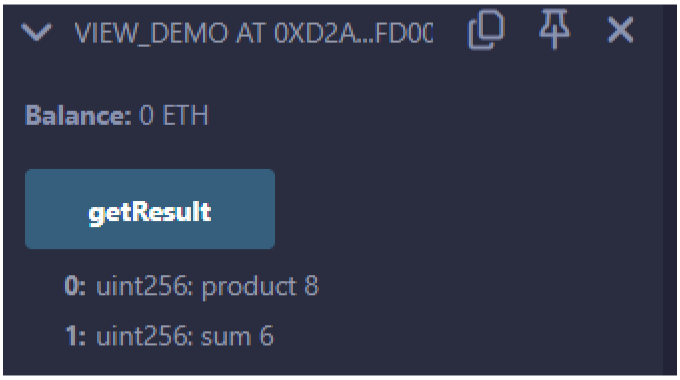
```
pragma solidity ^0.8.17;

contract Addition {

    int public input1;
    int public input2;

    function setInputs(int _input1, int _input2) public {
        input1 = _input1;
        input2 = _input2;
    }

    function additions() public view returns(int) {
        return input1 + input2;
    }

    function subtract() public view returns(int) {
        return input1 - input2;
    }
}
```



- **view functions**

Code:

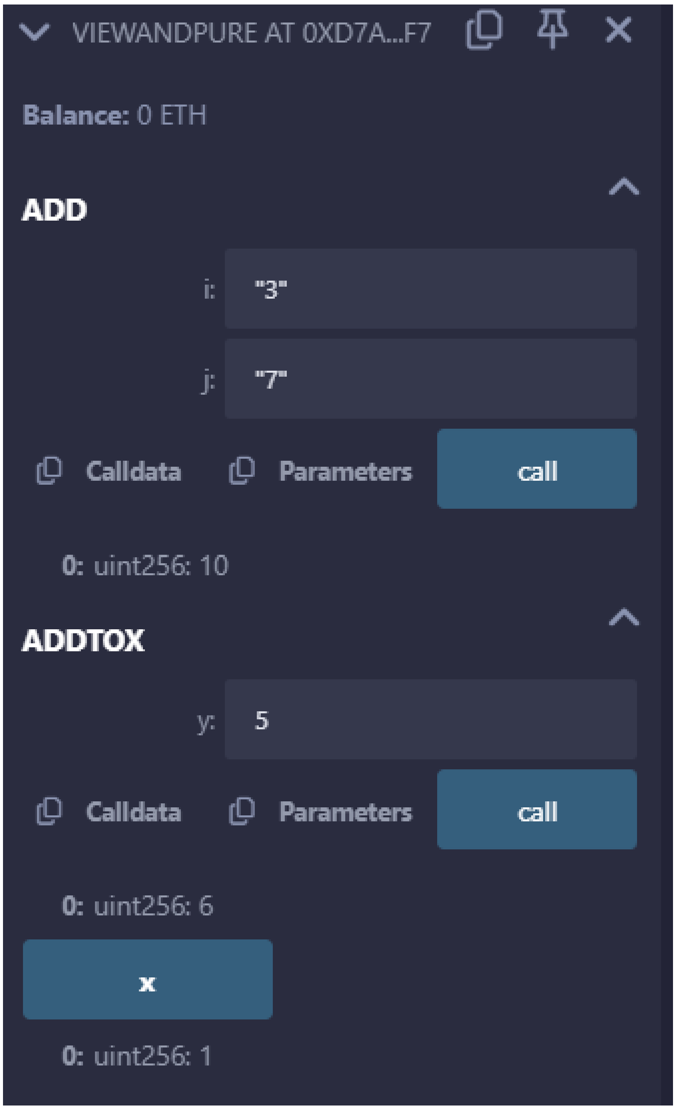
```
pragma solidity ^0.8.0;

contract view_demo
{
   uint256 num1 = 2;
   uint256 num2 = 4;

   function getResult() public view returns (uint256 product, uint256 sum) {
      product = num1 * num2;
      sum = num1 + num2;
   }
}
```



- **view and pure functions**

  Code:

```
pragma solidity ^0.8.3;

contract ViewAndPure {
   uint public x = 1;

   // Promise not to modify the state.
   function addToX(uint y) public view returns (uint) {
      return x + y;
   }

   // Promise not to modify or read from the state.
   function add(uint i, uint j) public pure returns (uint) {
      return i + j;
   }
}
```



- fallback function
  Code:

```
pragma solidity ^0.8.17;

contract fallbackfn {

   // Event to log details when fallback or receive function is called
   event Log(string func, address sender, uint value, bytes data);

   // Fallback function to handle calls to the contract with data or no matching function
   fallback() external payable {
      emit Log("fallback", msg.sender, msg.value, msg.data); // Emit log with details
   }

   // Receive function to handle plain ether transfers
   receive() external payable {
      emit Log("receive", msg.sender, msg.value, ""); // Emit log with details (msg.data is
empty)
```

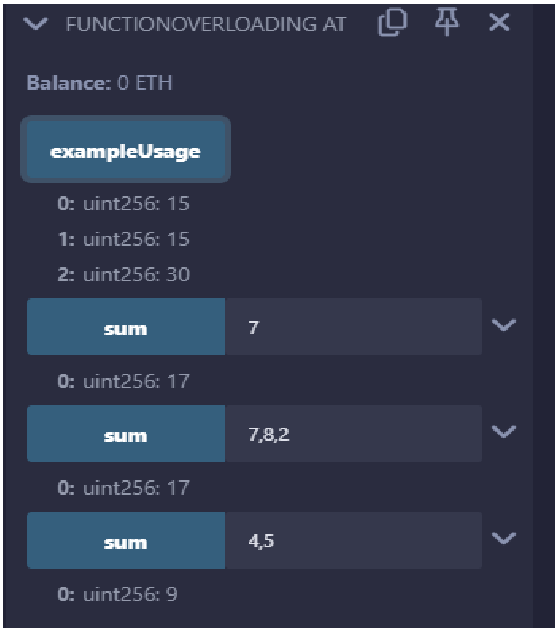
```
        //msg.data is empty hence no need to specify it and mark it as empty string

    }
}
```

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

- function overloading

Code:

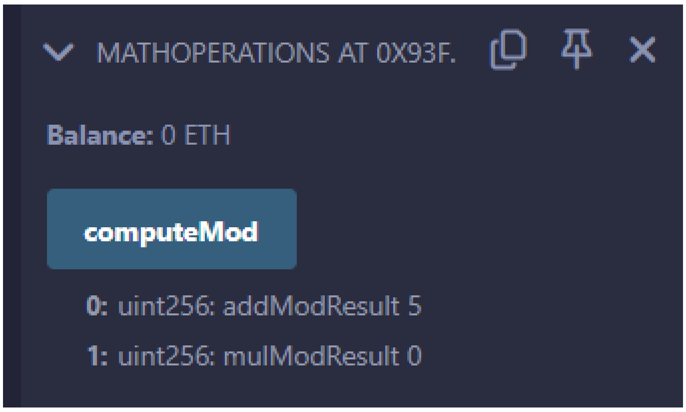
```
pragma solidity ^0.8.17;

contract FunctionOverloading {
    // Function with one parameter
    function sum(uint a) public pure returns (uint) {
        return a + 10;
    }

    // Overloaded function with two parameters
    function sum(uint a, uint b) public pure returns (uint) {
        return a + b;
    }
    // Overloaded function with three parameters
    function sum(uint a, uint b, uint c) public pure returns (uint) {
        return a + b + c;
    }
    // Examples of calling overloaded functions
    function exampleUsage() public pure returns (uint, uint, uint) {
        uint result1 = sum(5);           // Calls the first sum function
        uint result2 = sum(5, 10);       // Calls the second sum function
        uint result3 = sum(5, 10, 15);  // Calls the third sum function
        return (result1, result2, result3);
    }
}
```



- **mathematical functions**

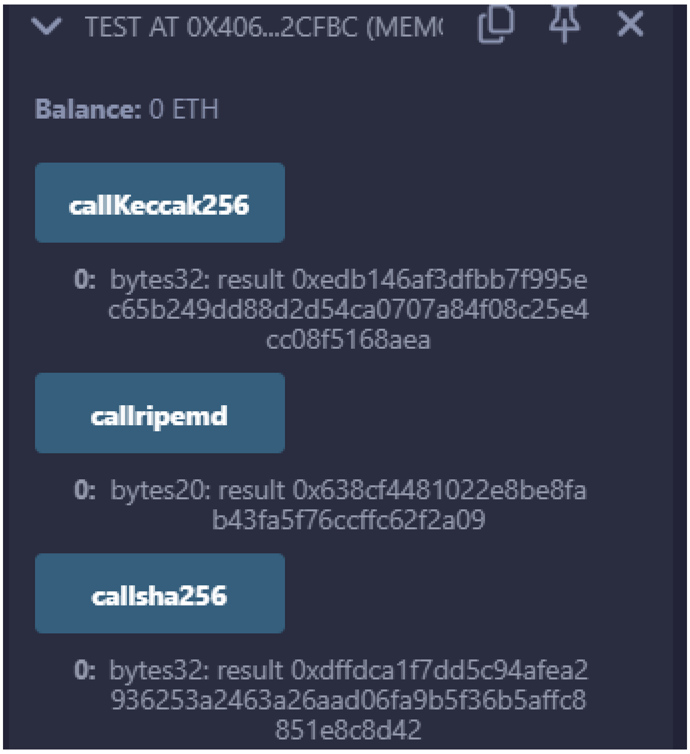
Code:

```
pragma solidity ^0.8.17;

contract MathOperations {
    // addMod computes (x + y) % k
    // mulMod computes (x * y) % k

    // Function to compute modular addition and multiplication
    // @return addModResult: Result of (x + y) % k
    // @return mulModResult: Result of (x * y) % k
    function computeMod() public pure returns (uint addModResult, uint mulModResult) {
        uint x = 3;
        uint y = 2;
        uint k = 6;
        addModResult = addmod(x, y, k); // Compute (x + y) % k
        mulModResult = mulmod(x, y, k); // Compute (x * y) % k
    }
}
```



- **cryptographic functions**
  Code:

```
pragma solidity ^0.8.17;
contract Test{
    function callKeccak256() public pure returns(bytes32 result){
        return keccak256("BLOCKCHAIN");
    }
    function callsha256() public pure returns(bytes32 result){
        return sha256("BLOCKCHAIN");
    }
    function callripemd() public pure returns (bytes20 result){
        return ripemd160("BLOCKCHAIN");
    }
}
```


TEST AT 0X406...2CFBC (MEM(
Balance: 0 ETH
callKeccak256
0: bytes32: result 0xedb146af3dfbb7f995ec65b249dd88d2d54ca0707a84f08c25e4cc08f5168aea
callripemd
0: bytes20: result 0x638cf4481022e8be8fab43fa5f76ccffc62f2a09
callsha256
0: bytes32: result 0xdffdca1f7dd5c94afea2936253a2463a26aad06fa9b5f36b5affc8851e8c8d42

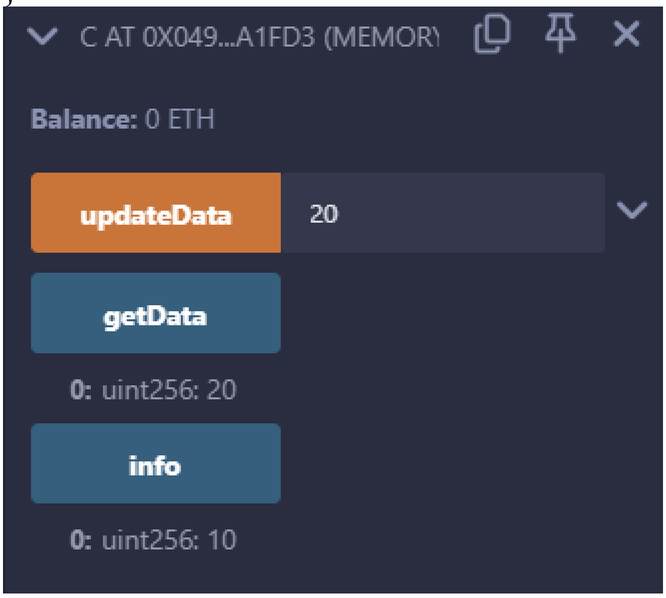
C. **Write a Solidity program that demonstrates various features including contracts, inheritance, constructors, abstract contracts, interfaces.**

- **Contracts, inheritance**

Code:

```
pragma solidity ^0.8.17;

contract C{

   uint private data;
   uint public info;

   constructor()  {
      info = 10;
      }

      function increment(uint a) private pure returns(uint){
         return a + 1;
      }

      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;
      }
}



contract D {

   function readData() public returns(uint) {
      C c = new C();
      c.updateData(7);
      return c.getData();
   }
}


contract E is C {

   uint private result;
   C private c;

   constructor()  {
```

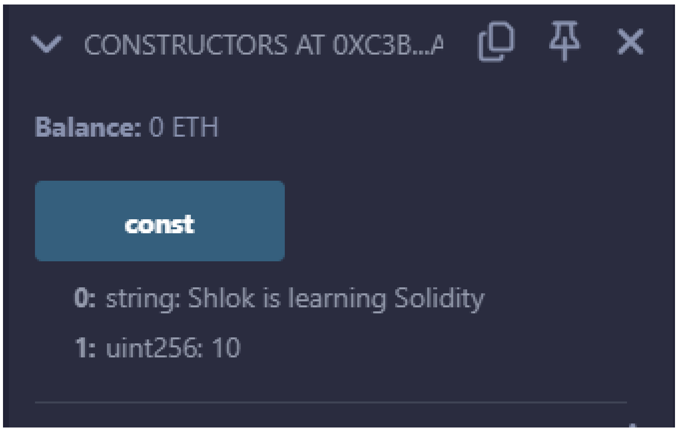
```
    c = new C();
  }

  function getComputedResult() public {
    result = compute(3, 6);
  }

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



- **Constructors**

Code:

```
pragma solidity ^0.8.17;

contract constructors{

    string str;
    uint amount;

    constructor(){
        str  = "Shlok is learning Solidity";
        amount = 10;
    }

    function const()public view returns(string memory,uint){
        return (str,amount);

    }

}
```



- **abstract contracts**

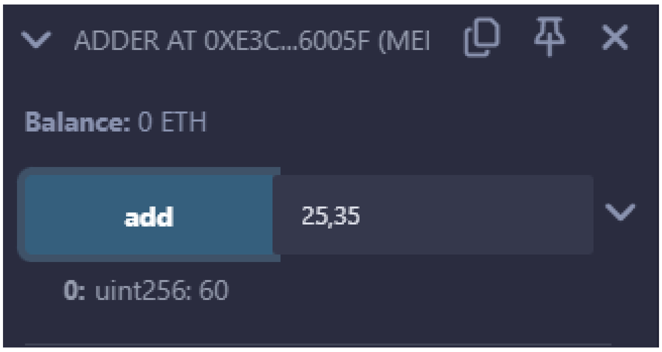
Code:

```
pragma solidity ^0.8.17;

abstract contract Main {
    // Define an abstract function that can be overridden
    function add(uint a, uint b) public virtual pure returns (uint);
}

contract Adder is Main {
    // Override the add function from the Main contract
    function add(uint a, uint b) public override pure returns (uint) {
        return a + b;
    }
}
```

- **interfaces**

Code:

```
pragma solidity ^0.8.17;

interface adder{
   function add(uint a, uint b)external pure returns(uint);
}

contract adderContract is adder{
   function add(uint a, uint b)external pure returns(uint){
      return a+b;
   }
}
```

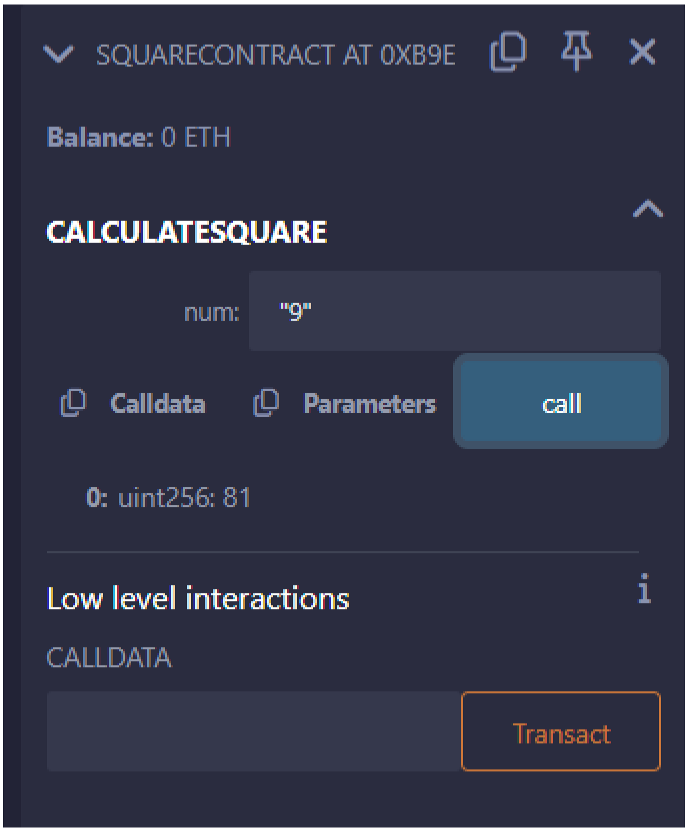


D. Write a Solidity program that demonstrates use of libraries, assembly, events, and error handling.

a. Libraries.
Code:

```
pragma solidity ^0.8.17;
library Search {
  function indexOf(uint[] storage self, uint value) internal view returns (uint) {
    for (uint i = 0; i < self.length; i++) {
      if (self[i] == value) {
        return i;
      }
    }
    return type(uint).max;
  }
}

contract Test {
  uint[] data;
  constructor() {
    data.push(1);
    data.push(2);
    data.push(3);
    data.push(4);
    data.push(5);
  }

  function isValuePresent() external view returns (uint) {
    uint value = 4;

    // Search if value is present in the array using Library function
    uint index = Search.indexOf(data, value);
    return index;
  }
}

library MathLibrary {
  function square(uint num) internal pure returns (uint) {
    return num * num;
  }
}
contract SquareContract {
  using MathLibrary for uint;
  function calculateSquare(uint num) external pure returns (uint) {
    return num.square();
  }
}
```

Output:

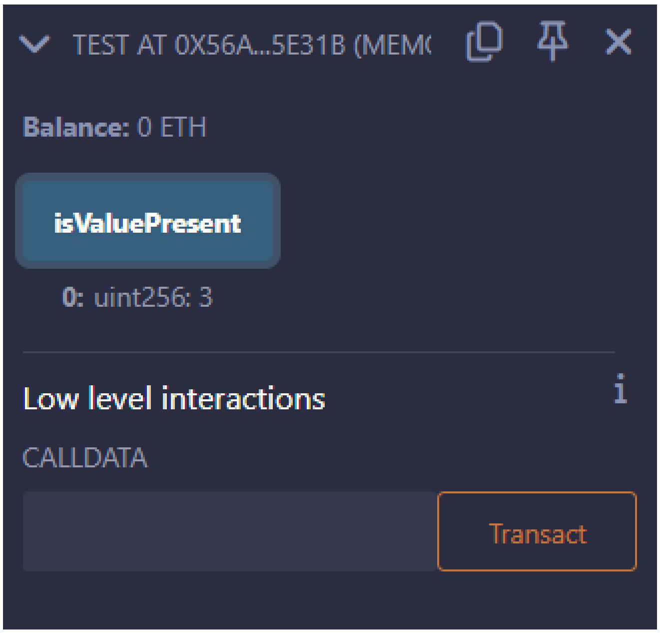


b. Assembly
Code:

```
pragma solidity ^0.8.17;

library Sum {
  function sumUsingInlineAssembly(uint[] memory _data) public pure returns (uint sum) {
    for (uint i = 0; i < _data.length; ++i) {
      assembly {
        // Load the value from memory at the current index
        let value := mload(add(add(_data, 0x20), mul(i, 0x20)))
        // Add the value to the sum
        sum := add(sum, value)
      }
    }
    // Return the calculated sum
    return sum;
  }
}
contract Test {
  uint[] data;
  constructor() {
    data.push(1);
    data.push(2);
    data.push(3);
    data.push(4);
    data.push(5);
  }
```

```
  function sum() external view returns (uint) {
    return Sum.sumUsingInlineAssembly(data);
  }
}
```

Output:

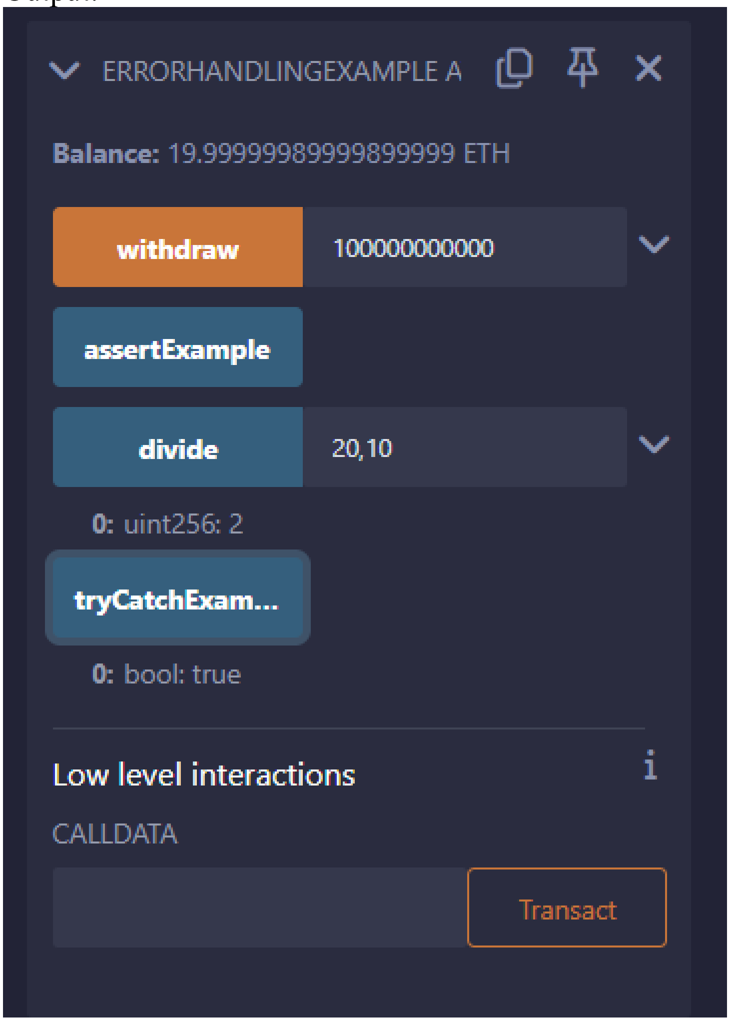


c. Error handling.
Code:

```
pragma solidity ^0.8.17;

contract ErrorHandlingExample {
    constructor() payable {
        // Allow the contract to receive Ether during deployment
    }

    function divide(uint256 numerator, uint256 denominator) external pure returns (uint256) {
        require(denominator != 0, "Division by zero is not allowed");
        return numerator / denominator;
    }

    function withdraw(uint256 amount) external {
        require(amount <= address(this).balance, "Insufficient balance");

        payable(msg.sender).transfer(amount);
    }

    function assertExample() external pure {
        uint256 x = 5;
        uint256 y = 10;
```

```
        assert(x < y);
    }

    function tryCatchExample() external view returns (bool) {
        try this.divide(10, 5) returns (uint256 result) {
            // Handle successful division
            return true;
        } catch Error(string memory errorMessage) {
            // Handle division error
            return false;
        }
    }
}
```

Output:



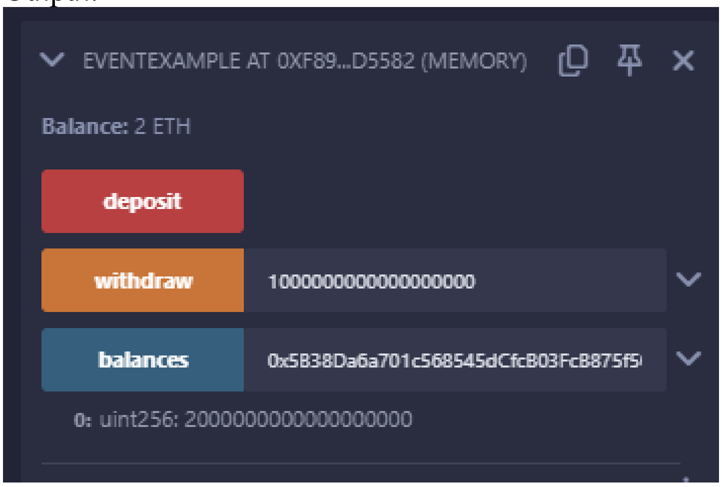
d. Events.

Code:

```
pragma solidity ^0.8.17;
contract EventExample {
    // Define an event
    event Deposit(address indexed from, uint256 amount);
    event Withdraw(address indexed to, uint256 amount);
    // Mapping to keep track of user balances
```

```
    mapping(address => uint256) public balances;
    // Function to deposit ether into the contract
    function deposit() public payable {
        require(msg.value > 0, "Must deposit more than 0 ether");
        // Update the balance
        balances[msg.sender] += msg.value;
        // Emit the Deposit event
        emit Deposit(msg.sender, msg.value);
    }
    // Function to withdraw ether from the contract
    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        // Update the balance
        balances[msg.sender] -= amount;
        // Transfer the ether
        payable(msg.sender).transfer(amount);
        // Emit the Withdraw event
        emit Withdraw(msg.sender, amount);
    }
}
```

Output:



Practical 4

Aim: Install and demonstrate use of hyperledger-Irhoa.

Hyperledger is an open source collaborative effort created to advance blockchain technologies. The Hyperledger organization has a number of projects (10 right now) for various blockchain solutions, such as smart contract engines, permissioned networks, querying for information inside a ledger, etc.

Iroha is a blockchain platform implementation meant for the simple modeling of financial transactions. So basically, if you have some asset with a quantity (maybe 100 forks and 200 spoons), you can easily model that on the blockchain, transfer those assets, see how many of those assets a certain person has, etc. Of course, this can be easily extended for various assets and quantities.

Iroha Terminology and Concepts
There are a few terms we need to know to get a good grasp on Iroha.
To begin learning the terms and concepts, we'll setup an example scenario, which we're going to model on the blockchain.
We own a farm, named "srcmakeFarm". We have some sheep, corn, a barn, and some other farm stuff on our land. We also have a few workers on our farm.
With that in mind, let's see the Iroha terminology that we need to model our farm.

1. **Asset** - Any countable commodity or value. It could be currency, like $US dollars$, Euros, bitcoins. It could be sheep, corn, windows. It can be anything that can be counted. We could have an asset named "sheep" on our blockchain.
2. **Account** - An entity that's able to perform a specified set of actions. For example, maybe one of our farmboys who feed the sheep has an account on our blockchain.
3. **Domain** - Assets and accounts are labeled by a domain, which is just a way of grouping them. For example, we could have a domain named "srcmakeFarm" that our "sheep" assets and "farmboy" account belong too.
4. **Permission** - Does an account have the privilege do perform a certain command? For example, to transfer sheeps from "srcmakeFarm" to another might require the "can_transfer" permission.
5. **Role** - A group of permissions. Perhaps we want the farmboy to be able to view how many sheep we have, but not be able to transfer the sheep to anyone else. In that case, we'd create a role for the farmboy with to view our asset count, but not to transfer them.
6. **Transaction** - A set of commands that are applied to the ledger. For example: 1) Add 20 "sheep" assets and 2) Remove 40 "sheep food" assets, could be some commands that form a transaction.
7. **Query** - Checking the state of data in the system. For example, a query to check how much sheep food "srcmakeFarm" currently has is valid.

Creating An Iroha Network
We're going to create a basic Iroha network.
Any Unix style terminal will work, but I've personally used Linux (Ubuntu 16.04).
The commands to install Docker on Ubuntu from the terminal are:

```
sudo apt-get install curl
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_relea
sudo apt-get update
apt-cache policy docker-ce
sudo apt-get install -y docker-ce
```

Next, we need to create a docker network. We'll name it "srcmake-iroha-network".

```
sudo docker network create srcmake-iroha-network
```

Next, we're going to add PostgreSQL to our network.

```
sudo docker run --name some-postgres \
-e POSTGRES_USER=postgres \
-e POSTGRES_PASSWORD=mysecretpassword \
-p 5432:5432 \
--network=srcmake-iroha-network \
-d postgres:9.5
```

Next, we'll create a volume of persistant storage named "blockstore" to store the blocks for our blockchain.

```
sudo docker volume create blockstore
```

Now, we need to configure Iroha on the network. Download the Iroha code from github. (And install git if you don't already have it.)

```
sudo apt-get install git
git clone -b develop https://github.com/hyperledger/iroh
```

We're going to use the Iroha configuration that's been prepared as an example to run the Iroha docker container.

```
sudo docker run -it --name iroha \
-p 50051:50051 \
-v $(pwd)/iroha/example:/opt/iroha_data \
-v blockstore:/tmp/block_store \
--network=srcmake-iroha-network \
--entrypoint=/bin/bash \
hyperledger/iroha:x86_64-develop-latest
```

Now we're going to actually run Iroha.

```
irohad --config config.docker --genesis_block genesis.bl
```

And now our Iroha blockchain is running in our terminal, so don't close it!

Interacting with our Iroha Network

So we have our Iroha network running, so let's make some transactions. If you look at the code for our genesis block, you can see that some commands were invoked to help get us started, so we're going to use those.

To interact with Iroha, we're going to use the command line tool.

Open a new terminal (don't close the one with our Iroha network!) and attach the docker container to our terminal.

```
sudo docker exec -it iroha /bin/bash
```

We should be inside the docker container's shell. Launch the iroha-cli tool and login as admin@test.

```
iroha-cli -account_name admin@test
```

```
root@f33e705c9721:/opt/iroha_data# iroha-cli -account_name admin@test
Welcome to Iroha-Cli.
Choose what to do:
1. New transaction (tx)
2. New query (qry)
3. New transaction status request (st)
> :
```

Type 1 and press enter to start a new transaction.

```
1
```

```
> : 1
Forming a new transactions, choose command to add:
1. Detach role from account (detach)
2. Add new role to account (apnd_role)
3. Create new role (crt_role)
4. Set account key/value detail (set_acc_kv)
5. Transfer Assets (tran_ast)
6. Grant permission over your account (grant_perm)
7. Subtract Assets Quantity from Account (sub_ast_qty)
8. Set Account Quorum (set_qrm)
9. Remove Signatory (rem_sign)
10. Create Domain (crt_dmn)
11. Revoke permission from account (revoke_perm)
12. Create Account (crt_acc)
13. Add Signatory to Account (add_sign)
14. Create Asset (crt_ast)
15. Add Peer to Iroha Network (add_peer)
16. Add Asset Quantity (add_ast_qty)
0. Back (b)
> :
```

There are a lot of commands that we can do. Let's try to model our farm a bit. Let's first create the domain.
Type 10 and create an domain with the id "srcmakeFarm". The default role name is role (list of permissions) the domain has, and we'll give it a "user" role that was created for us as part of this example.

```
10
srcmakeFarm
user
```

```
> : 10
Domain Id: srcmakeFarm
Default Role name: user
Command is formed. Choose what to do:
1. Add one more command to the transaction (add)
2. Send to Iroha peer (send)
3. Go back and start a new transaction (b)
4. Save as json file (save)
> : 1
```

Next, let's add some sheep to our farm. Type 1 to add one more command to the transaction. Then type 14 to create the sheep asset. It belongs to the domain srcmakeFarm, and the precision is 0 (meaning we don't allow decimals. Sheeps only come in whole numbers.)

```
1
14
sheep
srcmakeFarm
0
2
```

```
> : 14
Asset name: sheep
Domain Id: srcmakeFarm
Asset precision: 0
Command is formed. Choose what to do:
1. Add one more command to the transaction (add)
2. Send to Iroha peer (send)
3. Go back and start a new transaction (b)
4. Save as json file (save)
> : 2
```

Our farm looks pretty good, so let's send this to Iroha peer (the network) by typing 2. The peer address is "localhost" and it's on port "50051".

```
localhost
50051
```

```
> : 2
Peer address (0.0.0.0): localhost
Peer port (50051): 50051
[2018-04-25 03:53:24.674649529][th:41][info] TransactionResponseHandler Transact
ion successfully sent
Congratulation, your transaction was accepted for processing.
Its hash is 0fc3abceb23dce55f6756ccea788f81b47d0b96d7abb7a81984ae9f1c9154b33
--------------------
Choose what to do:
1. New transaction (tx)
2. New query (qry)
3. New transaction status request (st)
> : 
```

```
root@a645aef9ea62: /opt/iroha_data
[2018-04-25 03:53:24.666927875][th:57][info] TxProcessor handle transaction
[2018-04-25 03:53:24.666985842][th:57][info] PCS propagate tx
[2018-04-25 03:53:24.667006281][th:57][info] OrderingGate propagate tx, account_
id:  account_id: admin@test
[2018-04-25 03:53:24.667025256][th:57][info] OrderingGate Propagate tx (on trans
port)
[2018-04-25 03:53:24.673036508][th:20][info] OrderingServiceTransportGrpc Orderi
ngServiceTransportGrpc::onTransaction
[2018-04-25 03:53:24.673704479][th:20][info] OrderingServiceImpl Queue size is 1
[2018-04-25 03:53:28.465926418][th:191][info] OrderingServiceImpl Start proposal
```

You can see we get a hash for the transaction, and back in our Iroha network terminal, stuff happens (because we pushed a transaction onto the blockchain). so we have our srcmakeFarm and created an asset named sheep. Let's add 50 sheep (and give it to admin@test since we haven't made our own account).

Type 1 to make another transaction. Type 16 to add some quantity to an asset. The account is "admin@test", the asset is "sheep#srcmakeFarm" (notice the #domain), there are 50 sheep, and the precision is 0. Type 2 and send it to the localhost at 50051.

```
1
16
admin@test
sheep#srcmakeFarm
50
0
2
localhost
50051
```

```
> : 16
Account Id: admin@test
Asset Id: sheep#srcmakeFarm
Amount to add (integer part): 50
Amount to add (precision): 0
50 0
Command is formed. Choose what to do:
1. Add one more command to the transaction (add)
2. Send to Iroha peer (send)
3. Go back and start a new transaction (b)
4. Save as json file (save)
> : 2
Peer address (0.0.0.0): localhost
Peer port (50051): 50051
[2018-04-25 04:03:08.653226959][th:41][info] TransactionResponseHandler Transact
ion successfully sent
Congratulation, your transaction was accepted for processing.
Its hash is 2ddd8a4fd0af0507572a2b3fb8b0682ce5f01f775cfbf70c69d6cf101437be0b
--------------------
Choose what to do:
1. New transaction (tx)
2. New query (qry)
3. New transaction status request (st)
> : 
```

Making A Query
Let's check how many sheep we have. Type 2 to make a new query this time.

```
2
```

```
Choose what to do:
1. New transaction (tx)
2. New query (qry)
3. New transaction status request (st)
> : 2
Choose query:
1. Get all permissions related to role (get_role_perm)
2. Get Transactions by transactions' hashes (get_tx)
3. Get information about asset (get_ast_info)
4. Get Account's Transactions (get_acc_tx)
5. Get all current roles in the system (get_roles)
6. Get Account's Signatories (get_acc_sign)
7. Get Account's Assets (get_acc_ast)
8. Get Account Information (get_acc)
0. Back (b)
> :
```

There are lots of query types. We're going to query a particular account's assets, so type 7.

The account that owns the sheep is "admin@test", the asset we need is "sheep#srcmakeFarm". Send the query request to "localhost" at port "50051".

```
7
admin@test
sheep#srcmakeFarm
1
localhost
50051
```

```
> : 7
Requested account Id: admin@test
Requested asset Id: sheep#srcmakeFarm
Query is formed. Choose what to do:
1. Send to Iroha peer (send)
2. Save as json file (save)
0. Back (b)
> : 1
Peer address (0.0.0.0): localhost
Peer port (50051): 50051
[2018-04-25 04:16:12.946124963][th:41][info] QueryResponseHandler [Account Assets]
[2018-04-25 04:16:12.946210132][th:41][info] QueryResponseHandler -Account Id:- admin@test
[2018-04-25 04:16:12.946229238][th:41][info] QueryResponseHandler -Asset Id- sheep#srcmakeFarm
[2018-04-25 04:16:12.946251209][th:41][info] QueryResponseHandler -Balance- 50
--------------------
```

PRACTICAL JOURNAL

in

Deep Learning

Submitted to

Laxman Devram Sonawane College, Kalyan (W) 421301

in partial fulfilment for the award of the degree of

Master of Science in Information Technology

(Affiliated to Mumbai University)

Submitted by

Rithik Panneerselvam Nadar

Under the guidance of

Dr. Priyanka Pawar

Department of Information Technology
Kalyan, Maharashtra

Academic Year 2024-25

The Kalyan Wholesale Merchants Education Society's

Laxman Devram Sonawane College,

Kalyan (W) 421301

Department of Information Technology
Masters of Science – Part II

Certificate

This is to certify that **Mr. Rithik Panneerselvam Nadar**, Seat number_______, studying in Masters of Science in Information Technology Part II, Semester IV has satisfactorily completed the practical of "**Deep Learning** " as prescribed by University of Mumbai, during the academic year 2024-25.

Subject In-charge	Coordinator In-charge	ExternalExaminer

College Seal

INDEX

SR No.	Practical List	Sign
1	**Introduction to TensorFlow**	
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 TensorFlow	
b.	Program to solve the XOR problem	
2	**Linear Regression**	
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.	
b.	Make predictions on new data points	
3	**Convolutional Neural Networks (Classification)**	
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. O	
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 : Introduction to TensorFlow

1-a1. Create tensors with different shapes and data types

Code :

```
import tensorflow as tf

# Create a scalar (0-D tensor)
scalar = tf.constant(42)
print("Scalar (0-D Tensor):", scalar)

# Create a vector (1-D tensor)
vector = tf.constant([1.5, 2.5, 3.5], dtype=tf.float32)
print("Vector (1-D Tensor):", vector)

# Create a matrix (2-D tensor)
matrix = tf.constant([[1, 2], [3, 4], [5, 6]], dtype=tf.int32)
print("Matrix (2-D Tensor):", matrix)

# Create a 3-D tensor
tensor_3d = tf.constant([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("3-D Tensor:", tensor_3d)

# Create a tensor of all zeros
zeros_tensor = tf.zeros([2, 3])
print("Zeros Tensor:", zeros_tensor)

# Create a tensor of all ones
ones_tensor = tf.ones([3, 2, 2])
print("Ones Tensor:", ones_tensor)
```

```
# Create a tensor with random values
random_tensor = tf.random.normal([2, 2], mean=0, stddev=1)
print("Random Tensor:", random_tensor)

# Get the shape and data type of a tensor
print("Shape of matrix:", matrix.shape)
print("Data type of vector:", vector.dtype)
```

Output :

```
Scalar (0-D Tensor): tf.Tensor(42, shape=(), dtype=int32)
Vector (1-D Tensor): tf.Tensor([1.5 2.5 3.5], shape=(3,), dtype=float32)
Matrix (2-D Tensor): tf.Tensor(
[[1 2]
 [3 4]
 [5 6]], shape=(3, 2), dtype=int32)
3-D Tensor: tf.Tensor(
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]], shape=(2, 2, 2), dtype=int32)
Zeros Tensor: tf.Tensor(
[[0. 0. 0.]
 [0. 0. 0.]], shape=(2, 3), dtype=float32)
Ones Tensor: tf.Tensor(
[[[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]], shape=(3, 2, 2), dtype=float32)
Random Tensor: tf.Tensor(
[[-1.2160594  0.9616411]
 [-1.099461   0.8020662]], shape=(2, 2), dtype=float32)
Shape of matrix: (3, 2)
Data type of vector: <dtype: 'float32'>
```

1-a2. Perform basic operations like addition, subtraction, multiplication, and division on tensors.

Code :

```
import tensorflow as tf

# Define two tensors
a = tf.constant([3, 6, 9], dtype=tf.int32)
b = tf.constant([2, 4, 6], dtype=tf.int32)

# Perform basic arithmetic operations
addition = tf.add(a, b)
subtraction = tf.subtract(a, b)
multiplication = tf.multiply(a, b)
division = tf.divide(a, b)

# Display the results
print("Tensor A:", a.numpy())
print("Tensor B:", b.numpy())
print("Addition:", addition.numpy())
print("Subtraction:", subtraction.numpy())
print("Multiplication:", multiplication.numpy())
print("Division:", division.numpy())

# More tensor operations
squared = tf.square(a)
sqrt_b = tf.sqrt(tf.cast(b, tf.float32))
dot_product = tf.tensordot(a, b, axes=1)
```

```
print("Squared A:", squared.numpy())
print("Square root of B:", sqrt_b.numpy())
print("Dot product of A and B:", dot_product.numpy())
```

Output :

```
Tensor A: [3 6 9]
Tensor B: [2 4 6]
Addition: [ 5 10 15]
Subtraction: [1 2 3]
Multiplication: [ 6 24 54]
Division: [1.5 1.5 1.5]
Squared A: [ 9 36 81]
Square root of B: [1.4142135 2.        2.4494898]
Dot product of A and B: 84
```

1-a3. Reshape, slice, and index tensors to extract specific elements or sections

Code :

```
import tensorflow as tf

# Create a sample tensor
original_tensor = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Original Tensor:\n", original_tensor.numpy())

# Reshape the tensor
reshaped_tensor = tf.reshape(original_tensor, (1, 9))
print("\nReshaped Tensor (1x9):\n", reshaped_tensor.numpy())

# Slice the tensor (Extract rows 1 and 2, columns 1 and 2)
sliced_tensor = original_tensor[1:3, 1:3]
print("\nSliced Tensor (Rows 1-2, Columns 1-2):\n", sliced_tensor.numpy())

# Indexing (Extract specific elements)
first_element = original_tensor[0, 0]
last_row = original_tensor[-1]
print("\nFirst Element (0,0):", first_element.numpy())
print("Last Row:", last_row.numpy())

# Use tf.gather to extract specific indices
gathered_elements = tf.gather(original_tensor, [0, 2], axis=0)
print("\nGathered Rows 0 and 2:\n", gathered_elements.numpy())

# Use tf.boolean_mask to extract elements with a condition
```

```
masked_elements = tf.boolean_mask(original_tensor, original_tensor > 4)
print("\nElements Greater than 4:\n", masked_elements.numpy())
```

Output :

```
Original Tensor:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Reshaped Tensor (1x9):
 [[1 2 3 4 5 6 7 8 9]]

Sliced Tensor (Rows 1-2, Columns 1-2):
 [[5 6]
 [8 9]]

First Element (0,0): 1
Last Row: [7 8 9]

Gathered Rows 0 and 2:
 [[1 2 3]
 [7 8 9]]

Elements Greater than 4:
 [5 6 7 8 9]
```

1-a4. Performing matrix multiplication and finding eigenvectors and eigenvalues using TensorFlow

Code :

```
import tensorflow as tf

import numpy as np


# Define two matrices

matrix_a = tf.constant([[3, 4], [5, 6]], dtype=tf.float32)

matrix_b = tf.constant([[7, 8], [9, 10]], dtype=tf.float32)


# Perform matrix multiplication

matrix_product = tf.matmul(matrix_a, matrix_b)


# Display the result of matrix multiplication

print("Matrix A:", matrix_a.numpy())

print("Matrix B:", matrix_b.numpy())

print("Matrix Product (A * B):", matrix_product.numpy())


# Finding eigenvalues and eigenvectors

matrix_c = tf.constant([[4, -2], [1, 1]], dtype=tf.float32)


# Convert TensorFlow tensor to NumPy array for eigenvalue computation

matrix_c_np = matrix_c.numpy()

eigenvalues, eigenvectors = np.linalg.eig(matrix_c_np)


# Display eigenvalues and eigenvectors

print("Matrix C:", matrix_c_np)

print("Eigenvalues:", eigenvalues)
```

print("Eigenvectors:\n", eigenvectors)

Output :

```
Matrix A: [[3. 4.]
 [5. 6.]]
Matrix B: [[ 7.  8.]
 [ 9. 10.]]
Matrix Product (A * B): [[ 57.  64.]
 [ 89. 100.]]
Matrix C: [[ 4. -2.]
 [ 1.  1.]]
Eigenvalues: [3. 2.]
Eigenvectors:
 [[0.8944272  0.70710677]
 [0.4472136  0.70710677]]
```

1b. Program to solve the XOR problem

Code :

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

# Define XOR inputs and outputs
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y = np.array([[0], [1], [1], [0]], dtype=np.float32)

# Define a simple neural network model
model = keras.Sequential([
    layers.Dense(4, activation='relu', input_shape=(2,)),
    layers.Dense(4, activation='relu'),
    layers.Dense(1, activation='sigmoid')])

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

# Train the model
history = model.fit(X, y, epochs=1000, verbose=0)

# Evaluate the model
loss, accuracy = model.evaluate(X, y)
print(f"Final Loss: {loss:.4f}, Accuracy: {accuracy:.4f}")

# Make predictions
predictions = model.predict(X)
```

```
print("\nPredictions:")
for i, p in enumerate(predictions):
    print(f"Input: {X[i]} => Predicted Output: {p[0]:.4f} => Rounded: {int(np.round(p[0]))}")
```

Output :

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step - accuracy: 0.7500 - loss: 0.5135
Final Loss: 0.5135, Accuracy: 0.7500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step

Predictions:
Input: [0. 0.] => Predicted Output: 0.5979 => Rounded: 1
Input: [0. 1.] => Predicted Output: 0.5975 => Rounded: 1
Input: [1. 0.] => Predicted Output: 0.5979 => Rounded: 1
Input: [1. 1.] => Predicted Output: 0.1074 => Rounded: 0
```

Practical 2 : Linear Regression

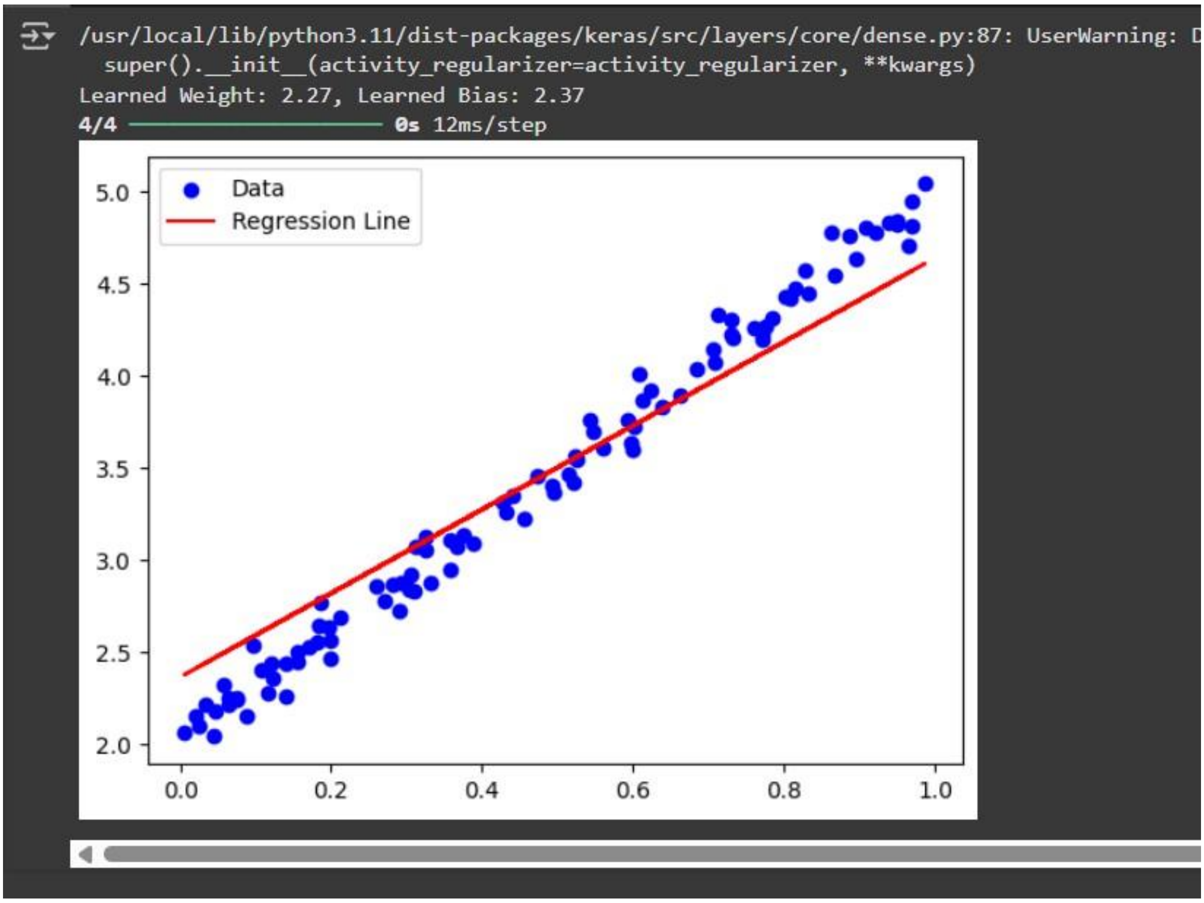
2-a1. Implement a simple linear regression model using TensorFlow's lowlevel API (or tf. keras).

Code :

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt


# Generate synthetic data

np.random.seed(42)

X = np.random.rand(100, 1).astype(np.float32)

y = 3 * X + 2 + np.random.normal(0, 0.1, (100, 1)).astype(np.float32)


# Define a simple linear regression model

model = tf.keras.Sequential([

    tf.keras.layers.Dense(1, input_shape=(1,))])


# Compile the model

model.compile(optimizer='sgd', loss='mse')


# Train the model

history = model.fit(X, y, epochs=200, verbose=0)


# Get the model's weights

W, b = model.layers[0].get_weights()

print(f"Learned Weight: {W[0][0]:.2f}, Learned Bias: {b[0]:.2f}")


# Make predictions
```

```
y_pred = model.predict(X)
# Plot the results
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.legend()
plt.show()
```

Output :



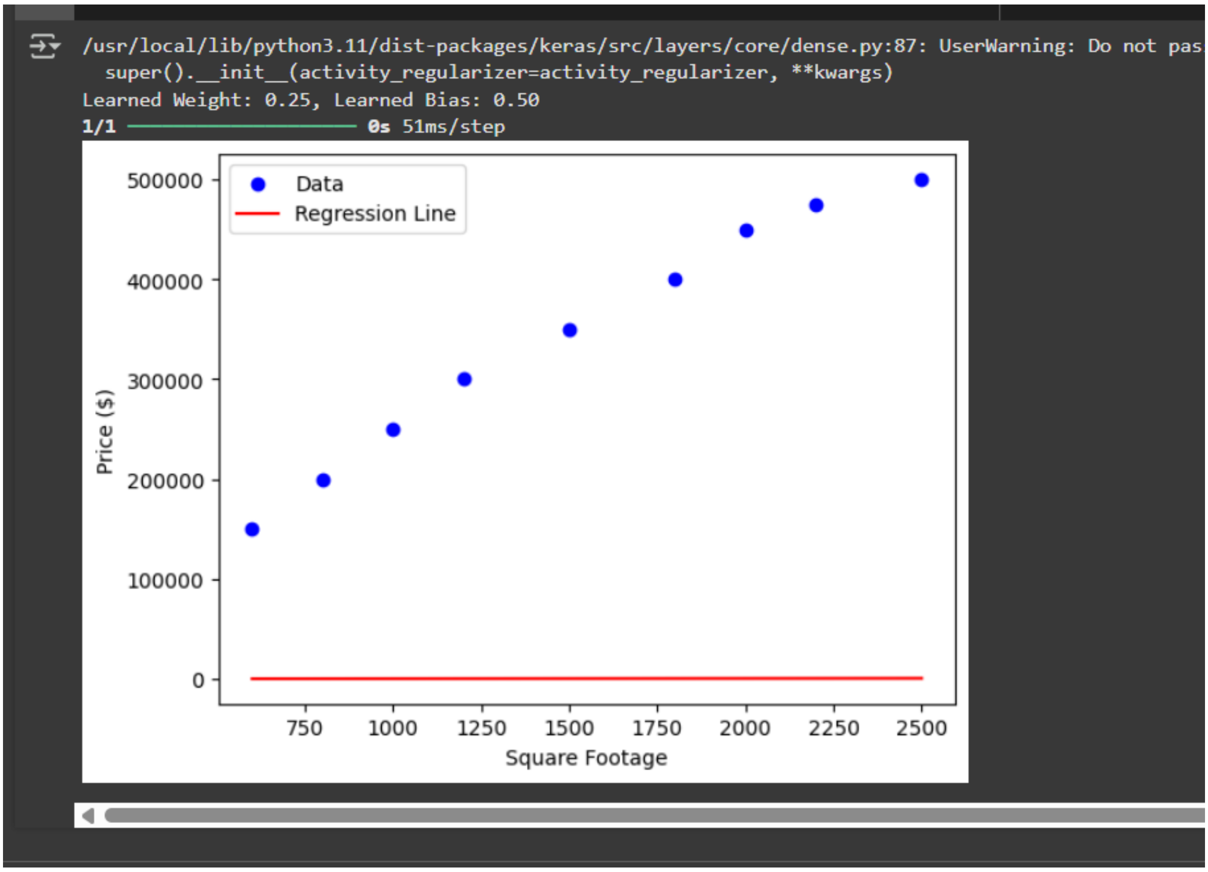
2-a2 Train the model on a toy dataset (e.g., housing prices vs. square footage).

Code :

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt


# Toy dataset: square footage vs. housing prices

square_feet = np.array([600, 800, 1000, 1200, 1500, 1800, 2000, 2200, 2500],
dtype=np.float32)

prices = np.array([150000, 200000, 250000, 300000, 350000, 400000, 450000, 475000,
500000], dtype=np.float32)


# Reshape data

X = square_feet.reshape(-1, 1)

y = prices.reshape(-1, 1)


# Define a simple linear regression model

model = tf.keras.Sequential([

    tf.keras.layers.Dense(1, input_shape=(1,))])


# Compile the model

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


# Train the model

history = model.fit(X, y, epochs=500, verbose=0)


# Get the model's weights

W, b = model.layers[0].get_weights()
```

```
print(f"Learned Weight: {W[0][0]:.2f}, Learned Bias: {b[0]:.2f}")
# Make predictions
y_pred = model.predict(X)

# Plot the results
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel('Square Footage')
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```

Output :



2-a3. Visualize the loss function and the learned linear relationship.

Code :

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt


# Toy dataset: square footage vs. housing prices

square_feet = np.array([600, 800, 1000, 1200, 1500, 1800, 2000, 2200, 2500], dtype=np.float32)

prices = np.array([150000, 200000, 250000, 300000, 350000, 400000, 450000, 475000, 500000], dtype=np.float32)


# Reshape data

X = square_feet.reshape(-1, 1)

y = prices.reshape(-1, 1)


# Define a simple linear regression model

model = tf.keras.Sequential([

   tf.keras.layers.Dense(1, input_shape=(1,))])


# Compile the model

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

# Train the model

history = model.fit(X, y, epochs=500, verbose=0)


# Get the model's weights

W, b = model.layers[0].get_weights()

print(f"Learned Weight: {W[0][0]:.2f}, Learned Bias: {b[0]:.2f}")


# Make predictions
```

y_pred = model.predict(X)

 # Plot the results

plt.figure(figsize=(12, 5))

 # Plot the data and regression line

plt.subplot(1, 2, 1)

plt.scatter(X, y, color='blue', label='Data')

plt.plot(X, y_pred, color='red', label='Regression Line')

plt.xlabel('Square Footage')

plt.ylabel('Price ($)')

plt.legend()

plt.title('Linear Regression Fit')

 # Plot the loss function

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], color='orange')

plt.xlabel('Epochs')

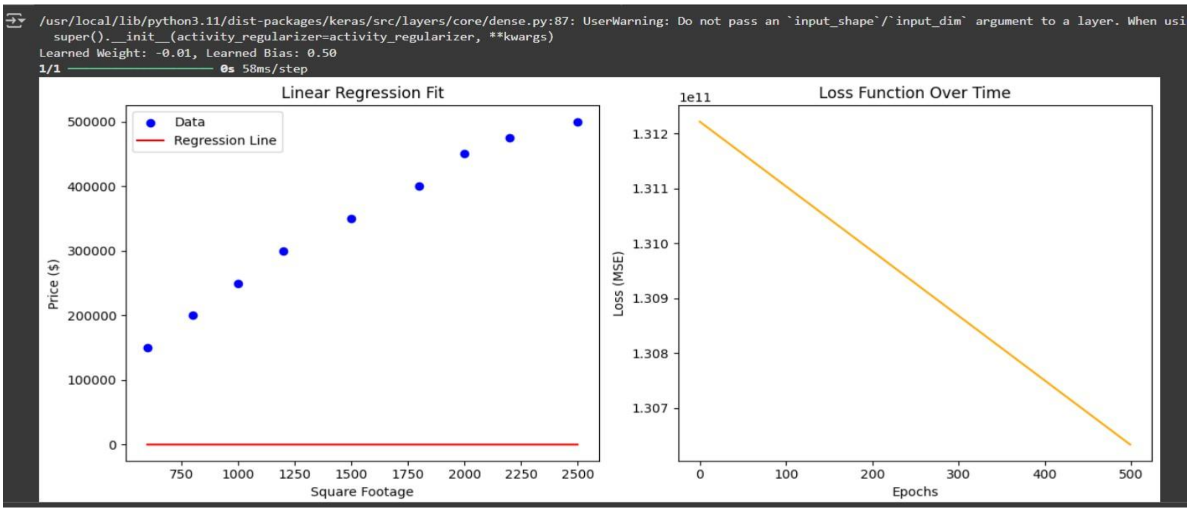
plt.ylabel('Loss (MSE)')

plt.title('Loss Function Over Time')

plt.tight_layout()

plt.show()

Output :



2-a4. Make predictions on new data points

Code :

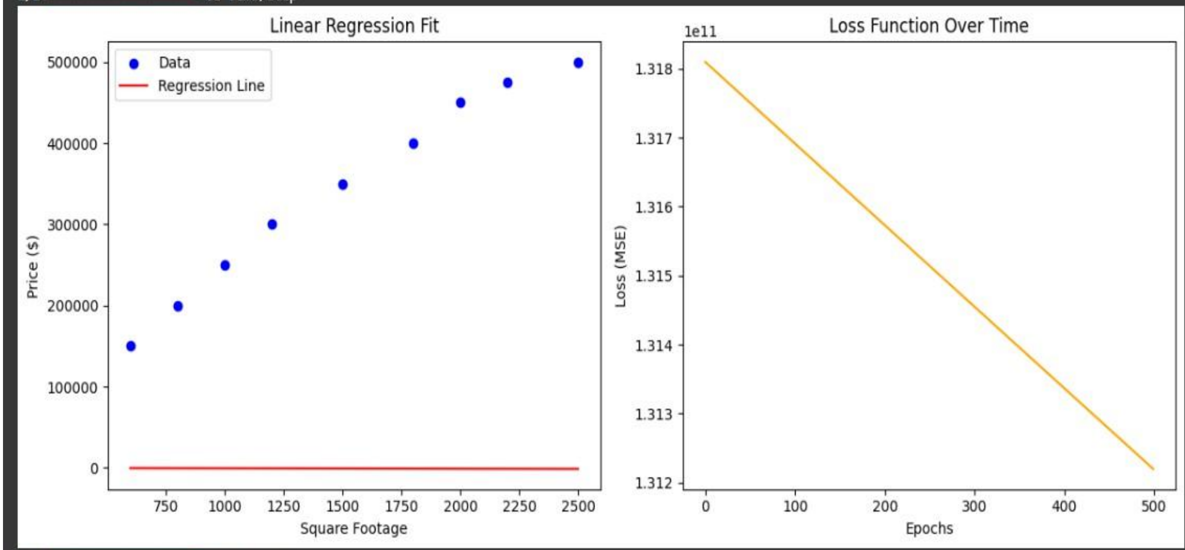
```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt


# Toy dataset: square footage vs. housing prices

square_feet = np.array([600, 800, 1000, 1200, 1500, 1800, 2000, 2200, 2500],
dtype=np.float32)

prices = np.array([150000, 200000, 250000, 300000, 350000, 400000, 450000, 475000,
500000], dtype=np.float32)


# Reshape data

X = square_feet.reshape(-1, 1)

y = prices.reshape(-1, 1)


# Define a simple linear regression model

model = tf.keras.Sequential([

   tf.keras.layers.Dense(1, input_shape=(1,))])


# Compile the model

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


# Train the model

history = model.fit(X, y, epochs=500, verbose=0)


# Get the model's weights

W, b = model.layers[0].get_weights()

print(f"Learned Weight: {W[0][0]:.2f}, Learned Bias: {b[0]:.2f}")
```

```
# Make predictions
y_pred = model.predict(X)

# Visualize the results
plt.figure(figsize=(12, 5))

# Plot the data and regression line
plt.subplot(1, 2, 1)
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel('Square Footage')
plt.ylabel('Price ($)')
plt.legend()
plt.title('Linear Regression Fit')

# Plot the loss function
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], color='orange')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.title('Loss Function Over Time')
plt.tight_layout()
plt.show()

# Make predictions on new data points
new_data = np.array([[1600], [2100], [3000]], dtype=np.float32)
new_predictions = model.predict(new_data)

print("\nPredictions on New Data:")
for sqft, price in zip(new_data.flatten(), new_predictions.flatten()):
```

print(f"Square Footage: {sqft}, Predicted Price: ${price:.2f}")

Output :

```
/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 Sequ
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Learned Weight: -0.50, Learned Bias: 0.50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
```

```
Predictions on New Data:
Square Footage: 1600.0, Predicted Price: $-805.47
Square Footage: 2100.0, Predicted Price: $-1057.33
Square Footage: 3000.0, Predicted Price: $-1510.69
```

Practical 3 : Convolutional Neural Networks (Classification)

3a. Implementing deep neural network for performing binary classification task

Code :

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt

# Load and preprocess data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# For binary classification, let's classify 'airplane' (class 0) vs 'automobile' (class 1)
# Reshape y_trrain & y_test to 1D arrays
y_train = y_train.reshape(-1)
y_test = y_test.reshape(-1)

X_train = X_train[(y_train == 0) | (y_train == 1)].astype('float32') / 255.0
X_test = X_test[(y_test == 0) | (y_test == 1)].astype('float32') / 255.0

y_train = y_train[(y_train == 0) | (y_train == 1)]
y_test = y_test[(y_test == 0) | (y_test == 1)]

# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')])
```

 # Compile the model

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

 # Train the model

```
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,
y_test), verbose=2)
```

 # Evaluate the model

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

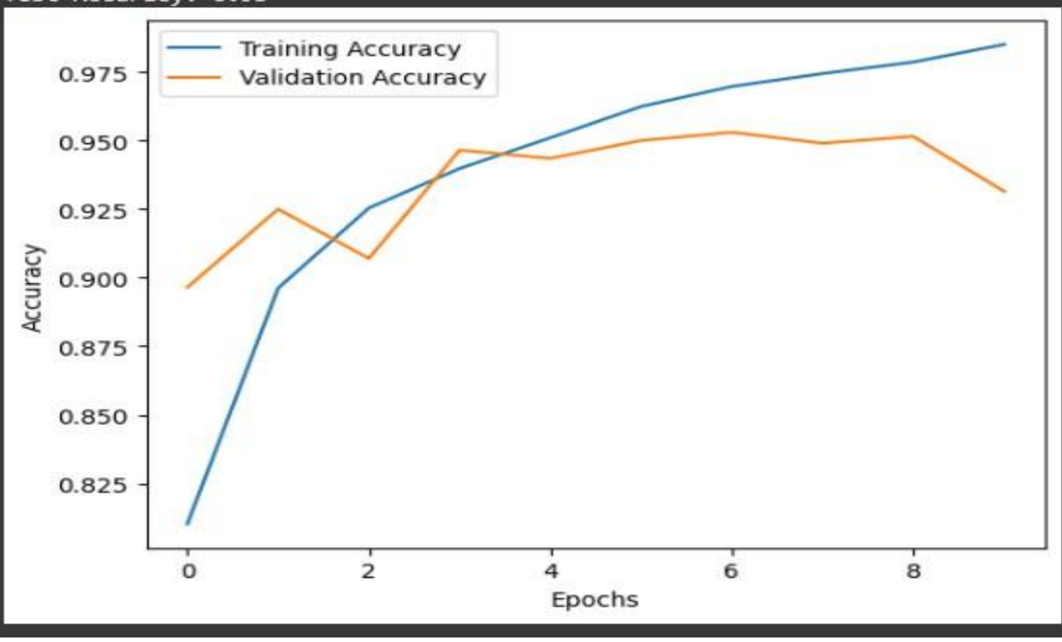
 # Plot training history

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

Output :

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
313/313 - 16s - 52ms/step - accuracy: 0.8103 - loss: 0.4080 - val_accuracy: 0.8965 - val_loss: 0.2718
Epoch 2/10
313/313 - 20s - 63ms/step - accuracy: 0.8962 - loss: 0.2507 - val_accuracy: 0.9250 - val_loss: 0.1876
Epoch 3/10
313/313 - 14s - 44ms/step - accuracy: 0.9254 - loss: 0.1894 - val_accuracy: 0.9070 - val_loss: 0.2266
Epoch 4/10
313/313 - 21s - 66ms/step - accuracy: 0.9398 - loss: 0.1529 - val_accuracy: 0.9465 - val_loss: 0.1405
Epoch 5/10
313/313 - 21s - 66ms/step - accuracy: 0.9510 - loss: 0.1233 - val_accuracy: 0.9435 - val_loss: 0.1526
Epoch 6/10
313/313 - 20s - 64ms/step - accuracy: 0.9624 - loss: 0.0994 - val_accuracy: 0.9500 - val_loss: 0.1584
Epoch 7/10
313/313 - 21s - 66ms/step - accuracy: 0.9697 - loss: 0.0846 - val_accuracy: 0.9530 - val_loss: 0.1345
Epoch 8/10
313/313 - 21s - 66ms/step - accuracy: 0.9744 - loss: 0.0689 - val_accuracy: 0.9490 - val_loss: 0.1546
Epoch 9/10
313/313 - 21s - 68ms/step - accuracy: 0.9786 - loss: 0.0595 - val_accuracy: 0.9515 - val_loss: 0.1363
Epoch 10/10
313/313 - 20s - 63ms/step - accuracy: 0.9850 - loss: 0.0421 - val_accuracy: 0.9315 - val_loss: 0.2261
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9363 - loss: 0.1993
Test Accuracy: 0.93
```

Test Accuracy: 0.93



3b. Using a deep feed-forward network with two hidden layers for performing multiclass classification and predicting the class.

Code :

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize the data
X_train = X_train.reshape(-1, 28 * 28).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28 * 28).astype('float32') / 255.0

# Define a deep feed-forward network model
model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(28 * 28,)),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')])
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,
y_test), verbose=2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

 # Make predictions on new data

predictions = model.predict(X_test[:5])

print("Predicted classes:", [tf.argmax(p).numpy() for p in predictions])

 # Plot training history

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

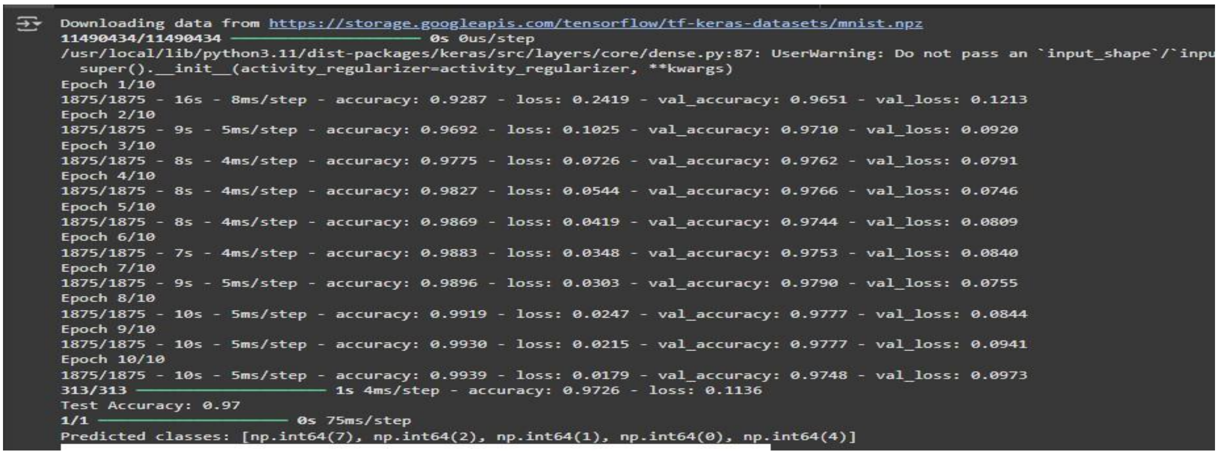
plt.xlabel('Epochs')

plt.ylabel('Accuracy')

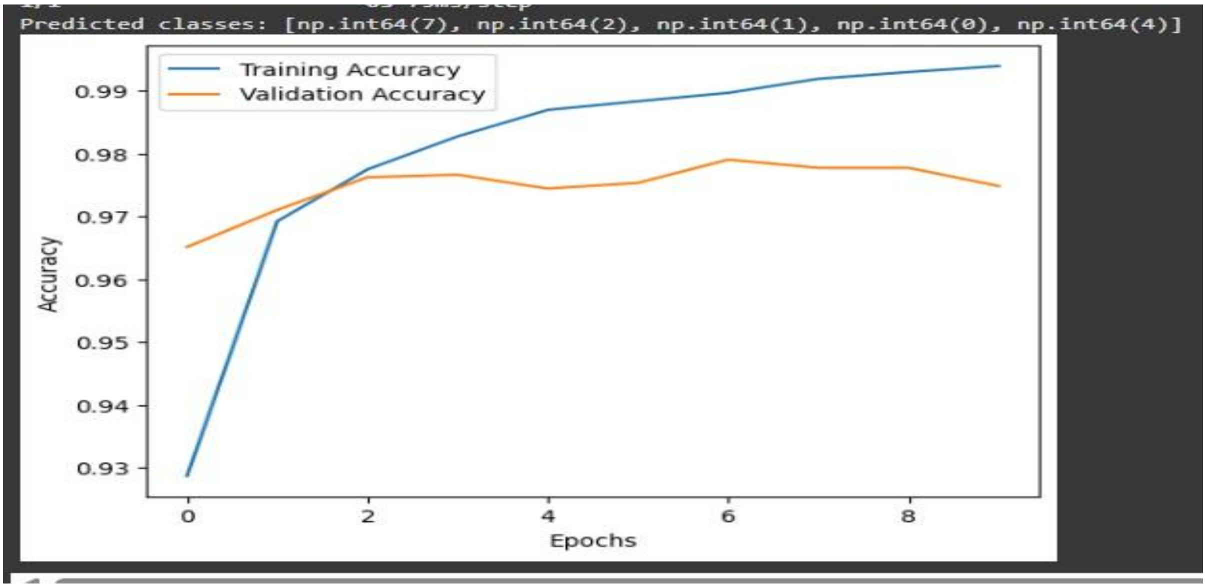
plt.legend()

plt.show()

Output :



```
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/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1875/1875 - 16s - 8ms/step - accuracy: 0.9287 - loss: 0.2419 - val_accuracy: 0.9651 - val_loss: 0.1213
Epoch 2/10
1875/1875 - 9s - 5ms/step - accuracy: 0.9692 - loss: 0.1025 - val_accuracy: 0.9710 - val_loss: 0.0920
Epoch 3/10
1875/1875 - 8s - 4ms/step - accuracy: 0.9775 - loss: 0.0726 - val_accuracy: 0.9762 - val_loss: 0.0791
Epoch 4/10
1875/1875 - 8s - 4ms/step - accuracy: 0.9827 - loss: 0.0544 - val_accuracy: 0.9766 - val_loss: 0.0746
Epoch 5/10
1875/1875 - 8s - 4ms/step - accuracy: 0.9869 - loss: 0.0419 - val_accuracy: 0.9744 - val_loss: 0.0809
Epoch 6/10
1875/1875 - 7s - 4ms/step - accuracy: 0.9883 - loss: 0.0348 - val_accuracy: 0.9753 - val_loss: 0.0840
Epoch 7/10
1875/1875 - 9s - 5ms/step - accuracy: 0.9896 - loss: 0.0303 - val_accuracy: 0.9790 - val_loss: 0.0755
Epoch 8/10
1875/1875 - 10s - 5ms/step - accuracy: 0.9919 - loss: 0.0247 - val_accuracy: 0.9777 - val_loss: 0.0844
Epoch 9/10
1875/1875 - 10s - 5ms/step - accuracy: 0.9930 - loss: 0.0215 - val_accuracy: 0.9777 - val_loss: 0.0941
Epoch 10/10
1875/1875 - 10s - 5ms/step - accuracy: 0.9939 - loss: 0.0179 - val_accuracy: 0.9748 - val_loss: 0.0973
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9726 - loss: 0.1136
Test Accuracy: 0.97
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
Predicted classes: [np.int64(7), np.int64(2), np.int64(1), np.int64(0), np.int64(4)]
```



Practical 4

Write a program to implement deep learning Techniques for image segmentation.

Code :

```
import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D,
concatenate

import numpy as np

import matplotlib.pyplot as plt


# Generate a small synthetic dataset

X_train = np.random.rand(10, 128, 128, 3)

y_train = np.random.randint(0, 2, (10, 128, 128, 1))

X_test = np.random.rand(2, 128, 128, 3)

y_test = np.random.randint(0, 2, (2, 128, 128, 1))


# Define a simple U-Net model

def build_simple_unet():

    inputs = Input(shape=(128, 128, 3))


    c1 = Conv2D(16, (3, 3), activation='relu', padding='same')(inputs)

    p1 = MaxPooling2D((2, 2))(c1)


    c2 = Conv2D(32, (3, 3), activation='relu', padding='same')(p1)

    p2 = MaxPooling2D((2, 2))(c2)


    c3 = Conv2D(64, (3, 3), activation='relu', padding='same')(p2)
```

```
    u1 = UpSampling2D((2, 2))(c3)
    u1 = concatenate([u1, c2])
    c4 = Conv2D(32, (3, 3), activation='relu', padding='same')(u1)

    u2 = UpSampling2D((2, 2))(c4)
    u2 = concatenate([u2, c1])
    c5 = Conv2D(16, (3, 3), activation='relu', padding='same')(u2)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c5)

    model = Model(inputs, outputs)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

model = build_simple_unet()
model.summary()

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=2, validation_data=(X_test, y_test))

# Test on a new image
def visualize_segmentation(model, image):
    pred_mask = model.predict(np.expand_dims(image, axis=0))[0]
    pred_mask = (pred_mask > 0.5).astype(np.uint8)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title('Original Image')
    plt.imshow(image)
```

```
plt.subplot(1, 2, 2)
plt.title('Predicted Mask')
plt.imshow(pred_mask.squeeze(), cmap='gray')
plt.show()
```

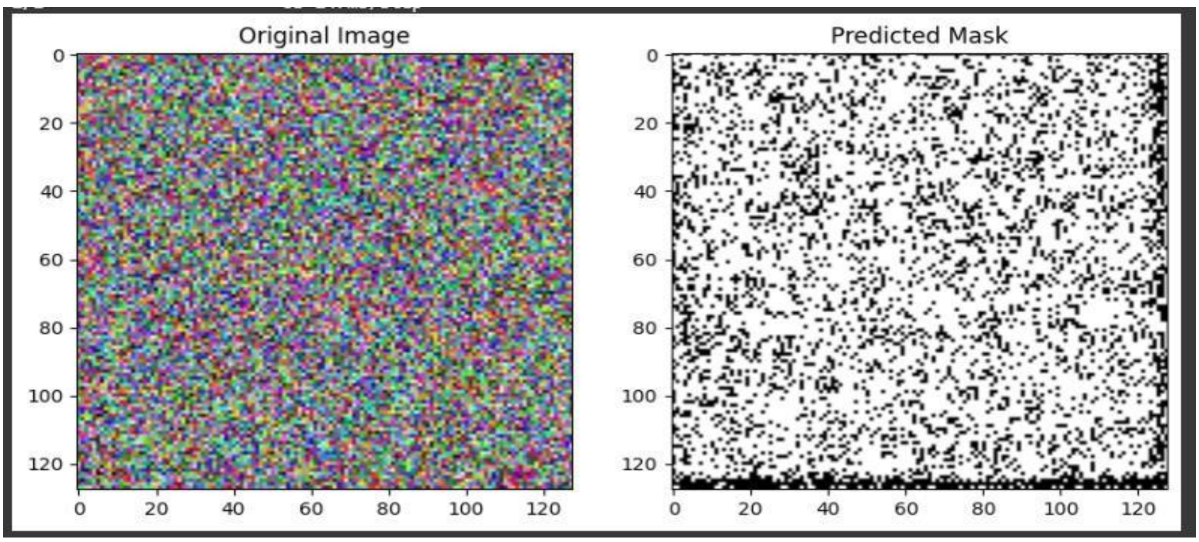
```
# Test on a random image
random_idx = np.random.randint(0, X_test.shape[0])
visualize_segmentation(model, X_test[random_idx])
```

Output :

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 128, 3)	0	-
conv2d (Conv2D)	(None, 128, 128, 16)	448	input_layer[0][0]
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4,640	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18,496	max_pooling2d_1[0][0]
up_sampling2d (UpSampling2D)	(None, 64, 64, 64)	0	conv2d_2[0][0]
concatenate (Concatenate)	(None, 64, 64, 96)	0	up_sampling2d[0][0], conv2d_1[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 32)	27,680	concatenate[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 128, 128, 32)	0	conv2d_3[0][0]
concatenate_1 (Concatenate)	(None, 128, 128, 48)	0	up_sampling2d_1[0][0], conv2d[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 16)	6,928	concatenate_1[0][0]
conv2d_5 (Conv2D)	(None, 128, 128, 1)	17	conv2d_4[0][0]

Total params: 58,209 (227.38 KB)
Trainable params: 58,209 (227.38 KB)
Non-trainable params: 0 (0.00 B)



Practical 5

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

Code :

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
import tensorflow as tf

# Define a small dataset
images = ["/content/Dog.jpg", "/content/cat.jpg", "/content/bike.jpg"]
captions = [
    "startseq a dog running in the park endseq",
    "startseq a cat sitting on a couch endseq",
    "startseq a person riding a bike endseq"
]

# Load VGG16 model for image feature extraction
base_model = VGG16(weights='imagenet')
model_vgg = Model(inputs=base_model.input, outputs=base_model.layers[-2].output)

def extract_features(image_path):
    img = load_img(image_path, target_size=(224, 224))
    img_array = img_to_array(img)
```

```python
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    return model_vgg.predict(img_array)

# Tokenize the captions
tokenizer = Tokenizer()
tokenizer.fit_on_texts(captions)
max_length = max(len(c.split()) for c in captions)
vocab_size = len(tokenizer.word_index) + 1

# Convert captions to sequences
sequences = tokenizer.texts_to_sequences(captions)
padded_captions = pad_sequences(sequences, maxlen=max_length, padding='post')

# Define the model
embedding_dim = 256
hidden_units = 256
image_input = Input(shape=(4096,))
image_output = Dense(embedding_dim, activation='relu')(image_input)
caption_input = Input(shape=(max_length,))
embedding = Embedding(vocab_size, embedding_dim, mask_zero=True)(caption_input)
lstm_out = LSTM(hidden_units)(embedding)
combined = add([image_output, lstm_out])
decoder_output = Dense(vocab_size, activation='softmax')(combined)
model = Model(inputs=[image_input, caption_input], outputs=decoder_output)
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Dummy training labels
y_train = np.zeros((3, vocab_size))
model.fit([np.random.rand(3, 4096), padded_captions], y_train, epochs=5, batch_size=1)
```

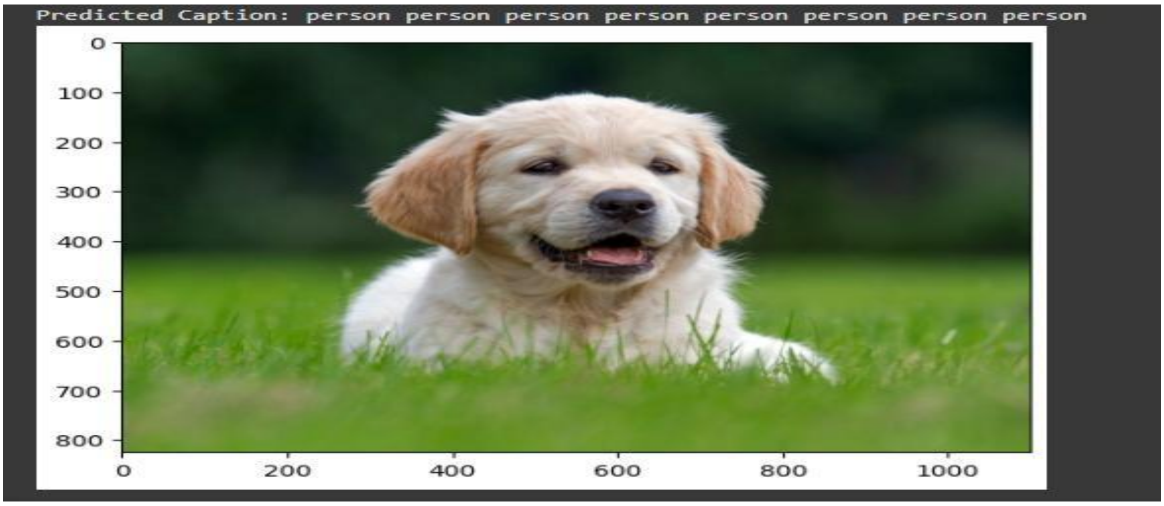
 # Generate a caption

```
def generate_caption(model, tokenizer, image_feature, max_length):
    in_text = 'startseq'
    for _ in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = np.argmax(model.predict([image_feature, sequence], verbose=0))
        word = tokenizer.index_word.get(yhat, None)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text.replace('startseq', '').replace('endseq', '').strip()
```

 # Test on a sample image

```
sample_image_path = images[0]
plt.imshow(load_img(sample_image_path))
sample_feature = extract_features(sample_image_path)
caption = generate_caption(model, tokenizer, sample_feature, max_length)
print("Predicted Caption:", caption)
```

Output :



Practical 6

Applying the Autoencoder algorithms for encoding real-world data

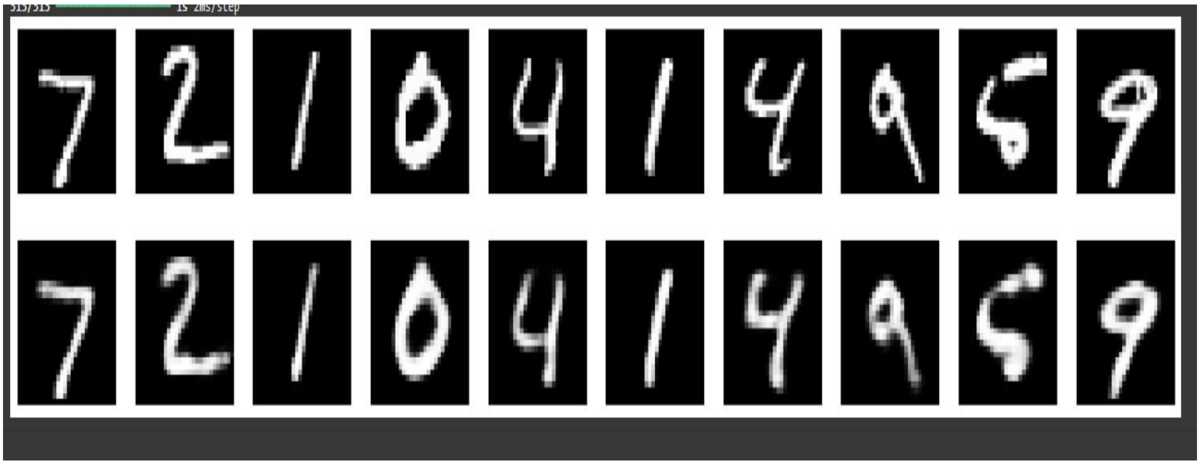
Code :

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

# Define the Autoencoder architecture
input_img = Input(shape=(28, 28, 1))

# Encoder
x = Flatten()(input_img)
x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
encoded = Dense(32, activation='relu')(x)

# Decoder
x = Dense(64, activation='relu')(encoded)
```

```
x = Dense(128, activation='relu')(x)
x = Dense(28 * 28, activation='sigmoid')(x)
decoded = Reshape((28, 28, 1))(x)

# Define the Autoencoder model
autoencoder = Model(input_img, decoded)

# Compile the model
autoencoder.compile(optimizer='adam',  loss='binary_crossentropy')

# Train the Autoencoder
autoencoder.fit(
    x_train, x_train,
    epochs=50,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test, x_test)
)

# Encode and decode some images
encoded_imgs = autoencoder.predict(x_test)

# Display original and reconstructed images
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
```

```
    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(encoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()
```

Output :



Practical 7

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

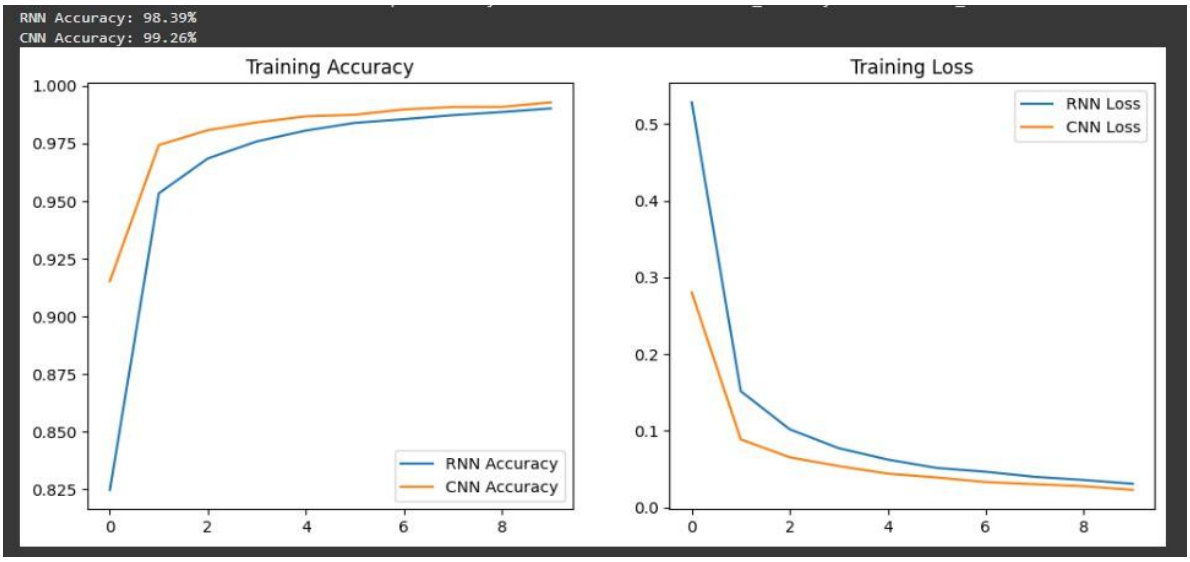
Code :

```
import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, LSTM, Conv2D, MaxPooling2D, Flatten,
Reshape, Dropout

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to_categorical


# Load and preprocess the MNIST dataset

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0


# Reshape data for CNN

x_train_cnn = x_train.reshape(-1, 28, 28, 1)

x_test_cnn = x_test.reshape(-1, 28, 28, 1)

# Reshape data for RNN

x_train_rnn = x_train.reshape(-1, 28, 28)

x_test_rnn = x_test.reshape(-1, 28, 28)


# One-hot encode the labels

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)
```

```
# Define RNN model
rnn_model = Sequential([
    LSTM(128, input_shape=(28, 28)),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')])

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

# Define CNN model
cnn_model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')])

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

# Train the RNN model
print("Training RNN model...")
rnn_history = rnn_model.fit(x_train_rnn, y_train, epochs=10, batch_size=128,
validation_data=(x_test_rnn, y_test))
# Train the CNN model
print("Training CNN model...")
cnn_history = cnn_model.fit(x_train_cnn, y_train, epochs=10, batch_size=128,
validation_data=(x_test_cnn, y_test))

# Evaluate the models
```

```
rnn_score = rnn_model.evaluate(x_test_rnn, y_test, verbose=0)
cnn_score = cnn_model.evaluate(x_test_cnn, y_test, verbose=0)

print(f"RNN Accuracy: {rnn_score[1] * 100:.2f}%")
print(f"CNN Accuracy: {cnn_score[1] * 100:.2f}%")

# Plot training loss and accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(rnn_history.history['accuracy'], label='RNN Accuracy')
plt.plot(cnn_history.history['accuracy'],  label='CNN  Accuracy')
plt.legend()
plt.title('Training Accuracy')

plt.subplot(1, 2, 2)
plt.plot(rnn_history.history['loss'], label='RNN Loss')
plt.plot(cnn_history.history['loss'], label='CNN Loss')
plt.legend()
plt.title('Training Loss')
plt.show()
```

Output :



Practical 8

Write a program to develop Autoencoders using MNIST Handwritten Digits

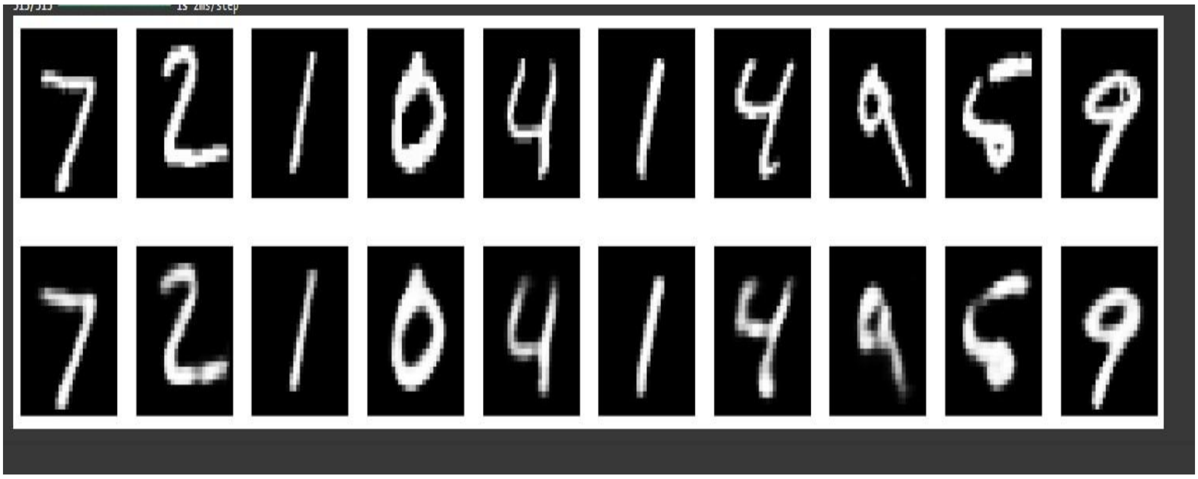
Code :

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

# Define the Autoencoder architecture
input_img = Input(shape=(28, 28, 1))

# Encoder
x = Flatten()(input_img)
x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
encoded = Dense(32, activation='relu')(x)

# Decoder
```

```
x = Dense(64, activation='relu')(encoded)
x = Dense(128, activation='relu')(x)
x = Dense(28 * 28, activation='sigmoid')(x)
decoded = Reshape((28, 28, 1))(x)

# Define the Autoencoder model
autoencoder = Model(input_img, decoded)

# Compile the model
autoencoder.compile(optimizer='adam',  loss='binary_crossentropy')

# Train the Autoencoder
autoencoder.fit(
    x_train, x_train,
    epochs=50,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test, x_test))

# Encode and decode some images
encoded_imgs = autoencoder.predict(x_test)

# Display original and reconstructed images
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
```

```
    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(encoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()
```

Output :



Practical 9

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 tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler

# Load Google stock price data
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "Google_Stock_Price_Train.csv"

# Load the latest version
df = kagglehub.load_dataset(
  KaggleDatasetAdapter.PANDAS,
  "vaibhavsxn/google-stock-prices-training-and-test-data",
  file_path,
  # Provide any additional arguments like
  # sql_query or pandas_kwargs. See the
  # documenation for more information:
```

```
 #
https://github.com/Kaggle/kagglehub/blob/main/README.md#kaggledatasetadapterpandas)

print("First 5 records:", df.head())

# Extract the 'Close' prices and preprocess
stock_prices = df['Close'].str.replace(',','',regex=True).astype(float).values
stock_prices = stock_prices.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_prices = scaler.fit_transform(stock_prices)

# Prepare data for RNN
sequence_length = 60
X, y = [], []
for i in range(len(scaled_prices) - sequence_length):
  X.append(scaled_prices[i:i+sequence_length])
  y.append(scaled_prices[i+sequence_length])
X, y = np.array(X), np.array(y)

# Reshape X for LSTM input
X = np.reshape(X, (X.shape[0], X.shape[1], 1))

# Build RNN model
model = Sequential([
  LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], 1)),
  Dropout(0.2),
  LSTM(units=50, return_sequences=False),
  Dropout(0.2),
  Dense(units=1)])

# Compile and train the model
```

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

model.fit(X, y, epochs=50, batch_size=32)

 # Make predictions

predicted_prices = model.predict(X)

predicted_prices = scaler.inverse_transform(predicted_prices)

 # Plot actual vs predicted stock prices

plt.plot(stock_prices[sequence_length:], color='blue', label='Actual Google Stock Price')

plt.plot(predicted_prices, color='red', label='Predicted Google Stock Price')

plt.title('Google Stock Price Prediction using RNN')

plt.xlabel('Time')

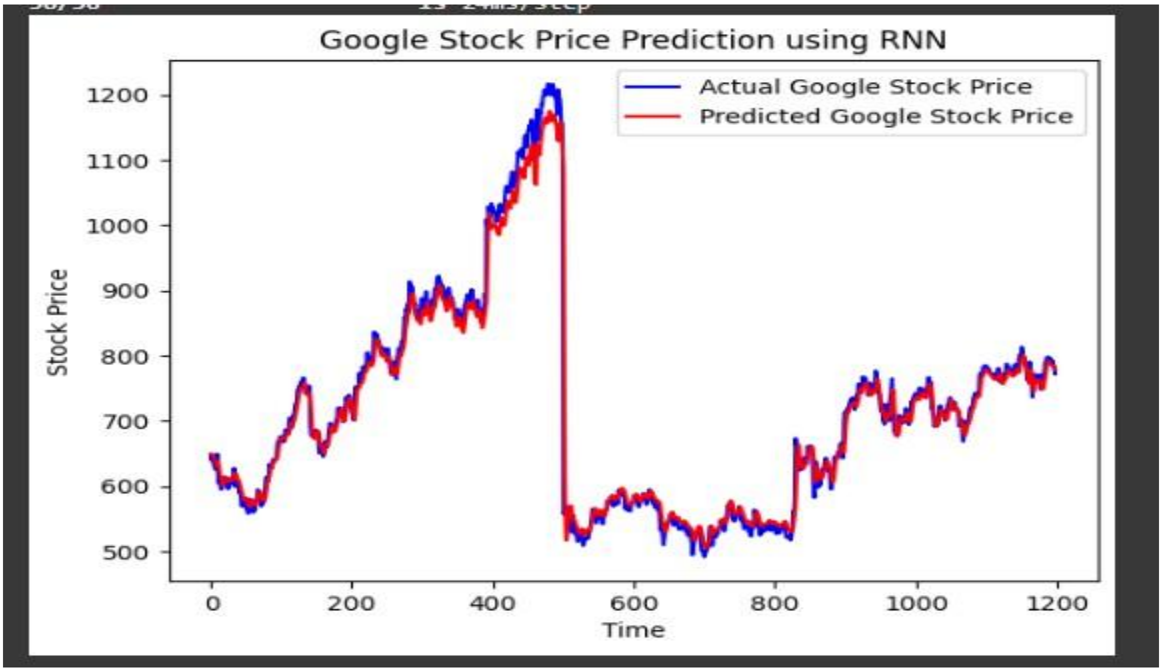
plt.ylabel('Stock Price')

plt.legend()

plt.show()

Output :

```
<ipython-input-3-6e5fd158cc1f>:17: DeprecationWarning: load_dataset is deprecated and will be removed in future version.
  df = kagglehub.load_dataset(
First 5 records:        Date    Open    High     Low   Close      Volume
0  1/3/2012  325.25  332.83  324.97  663.59   7,380,500
1  1/4/2012  331.27  333.87  329.08  666.45   5,749,400
2  1/5/2012  329.83  330.75  326.89  657.21   6,590,300
3  1/6/2012  328.34  328.77  323.68  648.24   5,405,900
4  1/9/2012  322.04  322.29  309.46  620.76  11,688,800
```



Practical 10

Applying Generative Adversarial Networks for image generation and unsupervised tasks.

Code :

```
import tensorflow as tf

from tensorflow.keras.layers import Dense, Reshape, Flatten, LeakyReLU, Dropout,
BatchNormalization, Conv2DTranspose, Conv2D

from tensorflow.keras.models import Sequential

import numpy as np

import matplotlib.pyplot as plt


# Define the Generator

def build_generator():

    model = Sequential()

    model.add(Dense(7 * 7 * 256, input_dim=100))

    model.add(LeakyReLU(0.2))

    model.add(Reshape((7, 7, 256)))


    model.add(Conv2DTranspose(128, kernel_size=4, strides=2, padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(0.2))


    model.add(Conv2DTranspose(64, kernel_size=4, strides=2, padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(0.2))


    model.add(Conv2D(1, kernel_size=7, padding='same', activation='tanh'))

    return model
```

 # Define the Discriminator

```
def build_discriminator():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=3, strides=2, padding='same', input_shape=(28, 28,
1)))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.3))

    model.add(Conv2D(128, kernel_size=3, strides=2, padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.3))

    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model
```

 # Compile the GAN

```
def build_gan(generator, discriminator):
    discriminator.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    discriminator.trainable = False
    model = Sequential([generator, discriminator])
    model.compile(loss='binary_crossentropy', optimizer='adam')
    return model
```

 # Load dataset (MNIST for simplicity)

```
(X_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
X_train = X_train / 127.5 - 1.0
X_train = np.expand_dims(X_train, axis=-1)
```

 # Training loop

```
def train_gan(gan, generator, discriminator, epochs=100, batch_size=64,
sample_interval=1000):
    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    for epoch in range(epochs):
        idx = np.random.randint(0, X_train.shape[0], batch_size)
        real_imgs = X_train[idx]

        noise = np.random.normal(0, 1, (batch_size, 100))
        fake_imgs = generator.predict(noise)

        d_loss_real = discriminator.train_on_batch(real_imgs, valid)
        d_loss_fake = discriminator.train_on_batch(fake_imgs, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        noise = np.random.normal(0, 1, (batch_size, 100))
        g_loss = gan.train_on_batch(noise, valid)

        if epoch % sample_interval == 0:
            print(f"Epoch {epoch}, D Loss: {d_loss[0]}, G Loss: {g_loss}")
            sample_images(generator)

# Generate and display images
def sample_images(generator, image_grid_rows=4, image_grid_columns=4):
    noise = np.random.normal(0, 1, (image_grid_rows * image_grid_columns, 100))
    gen_imgs = generator.predict(noise)
    gen_imgs = 0.5 * gen_imgs + 0.5

    fig, axs = plt.subplots(image_grid_rows, image_grid_columns, figsize=(4, 4), sharex=True,
sharey=True)
```

```
    count = 0
    for i in range(image_grid_rows):
        for j in range(image_grid_columns):
            axs[i, j].imshow(gen_imgs[count, :, :, 0], cmap='gray')
            axs[i, j].axis('off')
            count += 1
    plt.show()

# Build and compile the GAN
generator = build_generator()
discriminator = build_discriminator()
gan = build_gan(generator, discriminator)

# Train the GAN
train_gan(gan, generator, discriminator)
```

Output :