Q)write a program for blockchain in python to create multiple transaction and display them.

import hashlib

import json

from time import time

class Transaction:

    def \_\_init\_\_(self, sender, recipient, amount):

        self.sender = sender

        self.recipient = recipient

        self.amount = amount

    def to\_dict(self):

        return {

            'sender': self.sender,

            'recipient': self.recipient,

            'amount': self.amount

        }

    def \_\_repr\_\_(self):

        return json.dumps(self.to\_dict())

class Block:

    def \_\_init\_\_(self, index, transactions, previous\_hash, nonce=0):

        self.index = index

        self.timestamp = time()

        self.transactions = transactions  # list of Transaction objects

        self.previous\_hash = previous\_hash

        self.nonce = nonce

        self.hash = self.compute\_hash()

    def compute\_hash(self):

        # Convert transactions to dictionaries before serialization

        transaction\_dicts = [t.to\_dict() for t in self.transactions]

        block\_dict = {

            'index': self.index,

            'timestamp': self.timestamp,

            'transactions': transaction\_dicts,  # Use list of dictionaries

            'previous\_hash': self.previous\_hash,

            'nonce': self.nonce

        }

        block\_string = json.dumps(block\_dict, sort\_keys=True)  # Serialize the dictionary

        return hashlib.sha256(block\_string.encode()).hexdigest()

    def \_\_repr\_\_(self):

        return json.dumps({

            'index': self.index,

            'timestamp': self.timestamp,

            'transactions': [transaction.to\_dict() for transaction in self.transactions],

            'previous\_hash': self.previous\_hash,

            'hash': self.hash,

            'nonce': self.nonce

        }, indent=4)

# ... (Rest of the code remains the same)

class Blockchain:

    def \_\_init\_\_(self):

        self.chain = []

        self.pending\_transactions = []

        self.create\_genesis\_block()

    def create\_genesis\_block(self):

        genesis\_block = Block(0, [], "0")

        self.chain.append(genesis\_block)

    def create\_block(self, nonce):

        block = Block(

            index=len(self.chain),

            transactions=self.pending\_transactions,

            previous\_hash=self.chain[-1].hash,

            nonce=nonce

        )

        self.pending\_transactions = []

        self.chain.append(block)

        return block

    def add\_transaction(self, sender, recipient, amount):

        transaction = Transaction(sender, recipient, amount)

        self.pending\_transactions.append(transaction)

    def last\_block(self):

        return self.chain[-1]

    def proof\_of\_work(self, block, difficulty=4):

        block.nonce = 0

        computed\_hash = block.compute\_hash()

        while not computed\_hash.startswith('0' \* difficulty):

            block.nonce += 1

            computed\_hash = block.compute\_hash()

        return computed\_hash

    def is\_chain\_valid(self):

        for i in range(1, len(self.chain)):

            current = self.chain[i]

            previous = self.chain[i - 1]

            # Check if current block's hash is correct

            if current.hash != current.compute\_hash():

                return False

            # Check if current block's previous hash matches the previous block's hash

            if current.previous\_hash != previous.hash:

                return False

        return True

    def display\_chain(self):

        for block in self.chain:

            print(block)

def main():

    blockchain = Blockchain()

    # Add some transactions

    blockchain.add\_transaction("Alice", "Bob", 10)

    blockchain.add\_transaction("Bob", "Charlie", 20)

    # Create a new block with pending transactions

    blockchain.create\_block(nonce=100)

    blockchain.add\_transaction("Charlie", "Dave", 30)

    blockchain.add\_transaction("Dave", "Eve", 40)

    # Create another block

    blockchain.create\_block(nonce=200)

    # Display the blockchain

    blockchain.display\_chain()

    # Check if the blockchain is valid

    print(f"Blockchain valid: {blockchain.is\_chain\_valid()}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()