Open framework : remix.ethereum.org

1)Write a smart contract to store and retrieve value.

// SPDX-License-Identifier: MITpragma solidity 0.8.24;contract store{ uint value; function storevalue()external view returns(uint){ return value; } function getvalue(uint \_value)external{ value=\_value; }}

2) Write a smart contract for basic mathematical operations ( + , - , / , \* , %).

// SPDX-License-Identifier: MITpragma solidity 0.8.24;contract Addition{ uint public a=5; uint public b=5; function addition() view public returns(uint) { uint c=a+b; return c; } function sub() view public returns(uint) { uint c=a-b; return c; } function mul() view public returns(uint) { uint c=a\*b; return c; } function div() view public returns(uint) { uint c=a/b; return c; }}

3) Write a smart contract for incrementing and decrementing a number using event handling.

// SPDX-License-Identifier: MITpragma solidity 0.8.24;contract counter{ uint public count=0; event increment(uint value); event decrement(uint value); function getcount()view public returns(uint) { return count; } function Increment()public { count+=1; emit increment(count); } function Decrement()public { count-=1; emit decrement(count); }}

4) Write a smart contract for bank deposit and withdraw functionality.

// SPDX-License-Identifier: MITpragma solidity 0.8.24;contract bank{ uint public balance=12000; function viewbalnce()view public returns(uint){ return balance; } function deposit(uint amt) public { balance+=amt; } function withdraw(uint cash)public{ balance-=cash; }}

5) Write a smart contract for Degree certificate.

// SPDX-License-Identifier: MITpragma solidity 0.8.24;contract degreecertificate { struct Certificate { string studentName; string degreeTitle; uint256 dateIssued; address issuer; } mapping (address => Certificate) certificates; function issueCertificate(address student, string memory name, string memory title, uint256 date) public { require(certificates[student].issuer == address(0), "Certificate already issued for this student"); certificates[student] = Certificate(name, title, date, msg.sender); } function getCertificate(address student) public view returns (string memory, string memory, uint256, address) { Certificate memory cert = certificates[student]; require(cert.issuer != address(0), "No certificate issued for this student"); return (cert.studentName, cert.degreeTitle, cert.dateIssued, cert.issuer); } }

6) Write a smart contract for storing and retrieving the information for smart contract.

7) Write a simple python program to create a block and create a blockchain and show previous timestamp and hashcode.

import hashlib

import time

class Block:

def \_init\_(self, index, previous\_hash, data):

self.index = index

self.timestamp = time.time()

self.data = data

self.previous\_hash = previous\_hash

self.hash = self.calculate\_hash()

def calculate\_hash(self):

return hashlib.sha256(f"{self.index}{self.timestamp}{self.data}{self.previous\_hash}".encode('utf-8')).hexdigest()

class Blockchain:

def \_init(self):

self.chain = [self.create\_genesis\_block()]

def create\_genesis\_block(self):

return Block(0, "0","Genesis Block")

def get\_last\_block(self):

return self.chain[-1]

def add\_block(self, new\_block):

new\_block.previous\_hash = self.get\_last\_block().hash

new\_block.hash = new\_block.calculate\_hash()

self.chain.append(new\_block)

# Example usage

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

blockchain = Blockchain()

#Add blocks to the blockchain

block1 = Block(1,"", "Block 1 Data")

blockchain.add\_block(block1)

block2 = Block(2,"","Block 2 Data")

blockchain.add\_block(block2)

block3 = Block(3,"","Block 3 Data")

blockchain.add\_block(block3)

# Print blockchain details

for block in blockchain.chain:

print(f"Block {block.index}")

print(f"Timestamp : {block.timestamp}")

print(f"Data: {block.data}")

print(f"Previous Hash: {block.previous\_hash}")

print(f"Hash: {block.hash}")

print("")