# DEPARTMENT OF INFORMATION TECHNOLOGY

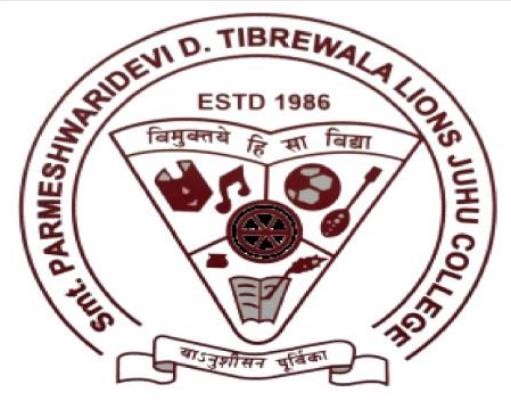
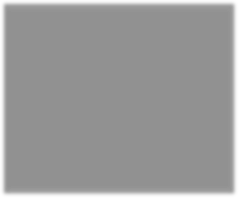
**SMT. PARMESHWARIDEVI DURGADUTT TIBREWALA**

# LIONS JUHU COLLEGE

**OF ARTS, COMMERE AND SCIENCE**

***Affiliated to University of Mumbai***

# J.B. NAGAR, ANDHERI (E), MUMBAI-400059



**Academic Year 2023-2024**

**BLOCKCHAIN**

***For***

**Semester IV**

# Submitted By:

**Mr. Abdul Rahim Karim Khan**

Msc.IT (Sem IV)

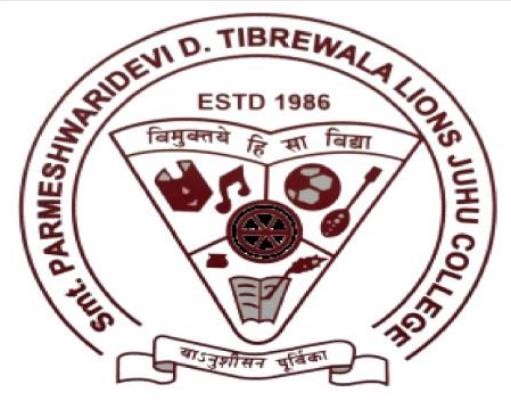
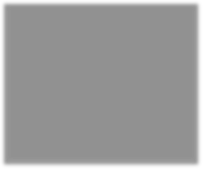
# SMT. PARMESHWARIDEVI DURGADUTT TIBREWALA LIONS JUHU COLLEGE

**OF ARTS, COMMERE AND SCIENCE**

***Affiliated to University of Mumbai***

# J.B. NAGAR, ANDHERI (E), MUMBAI-400059

**DEPARTMENT OF INFORMATION TECHNOLOGY**



# Certificate of Approval

This is to certify that practical entitled **“BLOCKCHAIN”**, Undertaken at **SMT.PARMESHWARIDEVI DURGADUTT TIBREWALA LIONS JUHU COLLEGE OF ARTS, COMMERECE & SCIENCE.** By **Mr. ABDUL RAHIM KARIM KHAN Seat No. \_\_\_\_\_\_\_\_\_\_\_\_**in partial fulfilment of **M.Sc. (IT) master degree (Semester IV)** Examination had not been submitted for any other examination and does not form of any other course undergone by the candidate. It is further certified that she has completed all required phases of the practical.

**Internal Examiner External Examiner**

**HOD / In-Charge / Coordinator Signature/**

**Principal/Stamp**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Name of the Practical** | **Page**  **No.** | **Date** | **Teacher’s**  **Signature** |
| **1** | Write the following programs for Blockchain in Python:  a. A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it. | 4 | 20-03-2024 |  |
|  | b. A transaction class to send and receive money and test it. | 7 | 20-03-2024 |  |
|  | c. Create multiple transactions and display them. | 9 | 27-03-2024 |  |
|  | d. Create a blockchain, a genesis block and execute it. | 14 | 27-03-2024 |  |
|  | e. Create a mining function and test it. | 16 | 03-04-2024 |  |
| **2** | Install and configure Go Ethereum and the Mist browser. | 18 | 10-04-2024 |  |
| **3** | Implement and demonstrate the use of the following in Solidity:  a. Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables. | 19 | 23-04-2024 |  |
|  | b. Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions. | 39 | 03-05-2024 |  |
| **4** | Implement and demonstrate the use of the following in Solidity:  a. Contracts, Inheritance, Interfaces. | 46 | 09-05-2024 |  |
|  | b. Libraries, Assembly, Error handling | 50 | 15-05-2024 |  |
| **5** | Create your own blockchain and demonstrate its use. | 53 | 28-05-2024 |  |

# PRACTICAL 1

**Write the following programs for Blockchain in Python**

**A. A simple client class that generates the private and public keys by using the builtin Python RSA algorithm and test it.**

**CODE:**

**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**

**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)**

**@property**

**def identity(self): return 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 == "Genesis":**

**identity = "Genesis" else:**

**identity = self.sender.identity**

**return collections.OrderedDict( { 'sender': identity,'recipient': self.recipient, 'value': self.value,'time' : self.time } )**

**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')**

**Sharik = Client() Dealer = Client()**

**t = Transaction( Sharik, Dealer.identity, 5.0**

**)**

**signature = t.sign\_transaction() print (signature)**

**class Block:**

**def init (self): self.verified\_transactions = []**

**self.previous\_block\_hash = "" self.Nonce = ""**

**block0 = Block() block0.previous\_block\_hash**

**= None**

**Nonce = None**

**t0 = Transaction ( "Genesis", Sharik.identity,**

**500.0**

**)**

**block0.verified\_transactions.append (t0) digest = hash (block0) last\_block\_hash = digest last\_block\_hash = "" digest = hash (block0) last\_block\_hash = digest**

**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) def dump\_blockchain (self):**

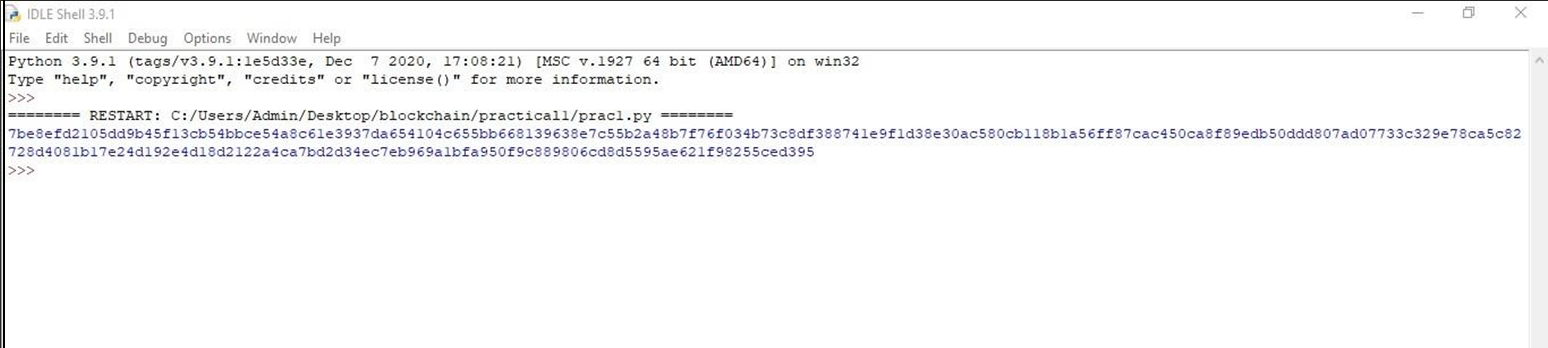
**print ("Number of blocks in the chain: " + str(len (self)))**

**for x in range (len(TPCoins)): block\_temp = TPCoins[x]**

**print ("block # " + str(x))**

**for transaction in block\_temp.verified\_transactions: display\_transaction (transaction)**

**OUTPUT:**

****

**B. A transaction class to send and receive money and test it.**

**Code:**

**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 collections**

**import Crypto**

**import Crypto.Random from Crypto.Hash**

**import SHA from Crypto.PublicKey**

**import RSA from Crypto.Signature import PKCS1\_v1\_5**

**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)**

**@property def identity(self):**

**return 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 == "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')**

**Sharik = Client() Sharik= Client()**

**t = Transaction( Sharik, sharik.identity, 5.0**

**)**

**signature = t.sign\_transaction()**

**print (signature)**

**OUTPUT:**

****

**C. Create multiple transactions and display them.**

**CODE:**

**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**

**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)**

**@property**

**def identity(self): return 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 == "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')**

**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 = [] Dinesh = Client() Ramesh = Client() Seema = Client() Vijay = Client()**

**t1 = Transaction( Dinesh, Ramesh.identity,15.0 )**

**t1.sign\_transaction() transactions.append(t1)**

**t2 = Transaction( Dinesh, Seema.identity, 6.0)**

**t2.sign\_transaction()**

**transactions.append(t2)**

**t3 = Transaction (Ramesh, Vijay.identity,2.0)**

**t3.sign\_transaction() transactions.append(t3)**

**t4 = Transaction( Seema, Ramesh.identity, 4.0)**

**t4.sign\_transaction() transactions.append(t4)**

**t5 = Transaction( Vijay, Seema.identity, 7.0)**

**t5.sign\_transaction()**

**transactions.append(t5)**

**t6 = Transaction (Ramesh, Seema.identity,3.0)**

**t6.sign\_transaction() transactions.append(t6)**

**t7 = Transaction( Seema, Dinesh.identity, 8.0)**

**t7.sign\_transaction() transactions.append(t7)**

**t8 = Transaction( Seema, Ramesh.identity, 1.0)**

**t8.sign\_transaction() transactions.append(t8)**

**t9 = Transaction( Vijay, Dinesh.identity, 5.0)**

**t9.sign\_transaction() transactions.append(t9)**

**t10 = Transaction( Vijay, Ramesh.identity,3.0)**

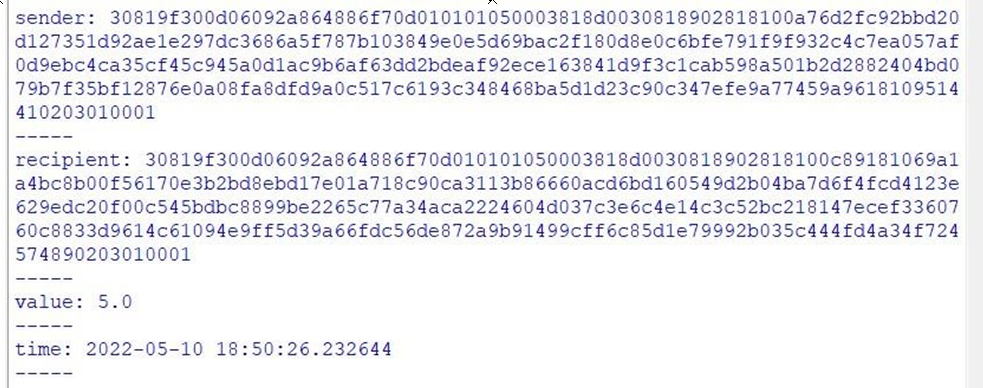
**t10.sign\_transaction() transactions.append(t10)**

**for transaction in transactions: display\_transaction (transaction) print (' ')**

**OUTPUT :**

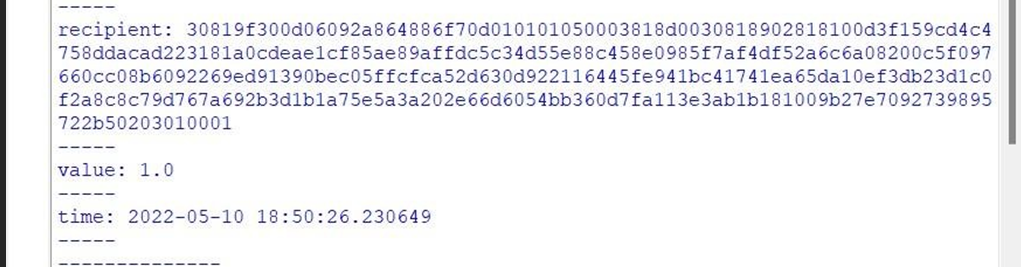




****

****

****

****

**D. Create a blockchain, a genesis block and execute it.**

**CODE:**

**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**

**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)**

**@property**

**def identity(self): return 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 == "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') 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 (' ')**

**class Block:**

**def init (self):**

**self.verified\_transactions = [] self.previous\_block\_hash = "" self.Nonce = "" last\_block\_hash = ""**

**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 ('=====================================')**

**Dinesh = Client()**

**t0 = Transaction ( "Genesis",Dinesh.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 = []**

**TPCoins.append (block0) dump\_blockchain(TPCoins)**

**OUTPUT :**

****

**E.** Create a mining function and test it

CODE:

**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**

**def sha256(message):**

**return hashlib.sha256(message.encode('ascii')).hexdigest()**

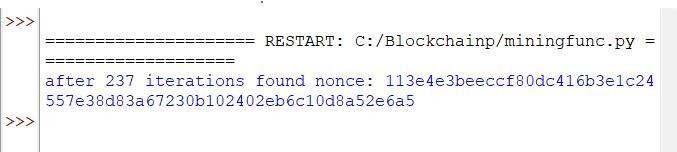
**def mine(message, difficulty=1):**

**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 ("test message", 2)**

**OUTPUT :**



**PRACTICAL 2**

**AIM: Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.**

**Installing GETH (Go Ethereum)**

**Step 1: Go to website https://geth.ethereum.org/downloads/ Step 2: From stable releases Geth 1.5.8 (kind = installer) Step 3: once downloaded run it then click next**

**Step 4: Select Geth and Development tools click next Step 5: Select location to install click next**

**Step 6: Once Installation is finished Click Close and its done**

**Installing Mist Browser**

**Step 1: https://github.com/ethereum/mist/releases**

**Step 2: Under Ethereum Wallet and Mist 0.8.9 - "The Wizard" download mist-installer-0-8- 9.exe**

**Step 3: For installation click, I agree -> next -> install**

**Run Mist**

**Step 1: Open the Mist from the start menu**

**Step 2: It will start downloading Blockchain data once you open it Step 3: Once it finishes downloading it is ready to use**

**Run Geth**

**Step 1: Open CMD**

**Step 2: Type GETH and press enter**

**Step 3: After it finishes loading press ctrl+c to exit the process.**

**Step 4: Now it's ready to use**

**PRACTICAL 3**

**AIM: Implement and demonstrate the use of the following in Solidity.**

1. **Variable, Operators, Loops, Decision Making, Strings, Arrays, Enums, Structs, Mappings, Conversions, Ether Units, Special Variables**

**VARIABLE CODE:**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**uint storedData;**

**// State variable constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(uint)**

**{**

**uint a = 1; // local variable**

**uint b = 2; uint result = a**

**+ b;**

**return storedData; //access the state variable**

**}**

**}**

**OUTPUT:**

**ARITHMETIC OPERATORS CODE**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**constructor()**

**public{}**

**function getResult()**

**public view returns(uint)**

**{**

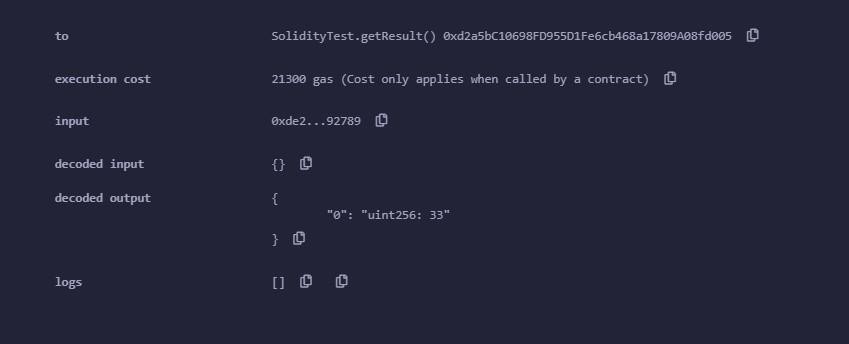
**uint a = 11; uint b = 22;**

**uint result = a + b; //arithmetic operation return result;**

**}**

**}**

**OUTPUT:**



**COMPARISON OPERATORS CODE:**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**uint storedData; constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{ uint a = 1; // local variable uint b = 2; uint result = a**

**+ b;**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory \_uintAsString)**

**{**

**if (\_i == 0)**

**{**

**//comparison operator return "0";**

**}**

**uint j = \_i;**

**uint len;**

**while (j != 0)**

**{**

**//comparison operator len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len);**

**uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

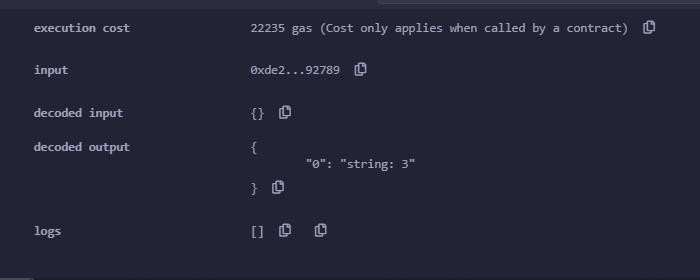
**}**

**return string(bstr);//access local variable**

**}**

**}**

**OUTPUT**



LOGICAL OPERATORS CODE

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**uint storedData; // State variable constructor() public{ storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 1; // local variable uint b = 2;**

**uint result = a**

**+ b;**

**return integerToString(storedData); //access the state variable**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0) { return "0";**

**} uint j = \_i;**

**uint len;**

**while (!(j == 0)) { //logical operator len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0) {**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

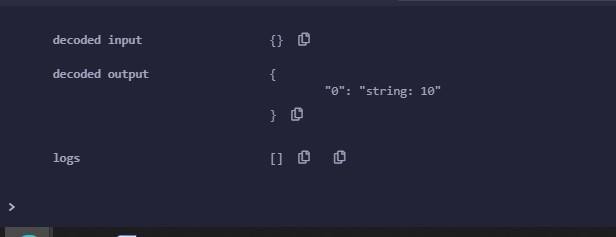
**}**

**return string(bstr);**

**}**

**}**

**OUTPUT:**



**BIT WISE OPERATORS CODE**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**uint storedData; constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 2; // local variable uint b = 2;**

**uint result = a & b; // bitwise operation**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0) { return "0";**

**}**

**uint j = \_i;**

**uint len;**

**while (j != 0) { len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

**}**

**return string(bstr);//access local variable**

**}**

**}**

**OUTPUT:**



**ASSIGNMENT OPERATORS CODE:**

**pragma solidity ^0.5.0;**

**contract SolidityTest { uint storedData; constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 1; uint b = 2; uint result = a + b;**

**return integerToString(storedData);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0) { return "0";**

**}**

**uint j = \_i; uint len; while (j != 0)**

**{**

**len++;**

**j /= 10; //assignment operation**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;//assignment operation**

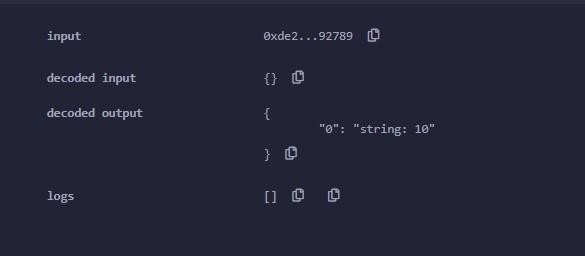
**}**

**return string(bstr); //access local variable**

**}**

**}**

**OUTPUT:**



**LOOP WHILE LOOP CODE:**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{ uint storedData; constructor() public**

**{ storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{ uint a = 10; uint b = 2; uint result = a + b;**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0) { return "0";**

**}**

**uint j = \_i;**

**uint len;**

**while (j != 0) { len++; j /=**

**10;**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0) { // while loop bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

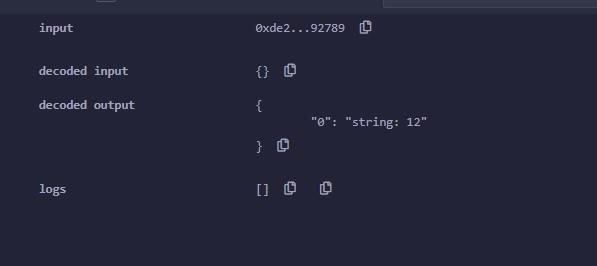
**}**

**return string(bstr);**

**}**

**}**

**OUTPUT:**



**DO WHILE LOOP CODE:**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{ uint storedData; constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 10; uint b = 2; uint result = a + b;**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory) {**

**if (\_i == 0)**

**{ return "0"; } uint j = \_i;**

**uint len;**

**while (j != 0) { len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**do { // do while loop**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

**}**

**while (\_i != 0);**

**return string(bstr);**

**}**

**}**

**OUTPUT:**

****

**FOR LOOP CODE:**

**pragma solidity ^0.5.0; contract SolidityTest**

**{**

**uint storedData;**

**constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{ uint a = 10;**

**uint b = 2;**

**uint result = a + b;**

**return integerToString(result);**

**function integerToString(uint \_i) internal pure**

**returns (string memory)**

**{**

**if (\_i == 0)**

**{**

**return "0";**

**}**

**uint j=0;**

**uint len;**

**for (j = \_i; j != 0; j /= 10)**

**{**

**//for loop example**

**len++;**

**}**

**bytes memory bstr = new bytes(len);**

**uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

**}**

**return string(bstr);//access local variable**

**}**

**}**

**OUTPUT:**



**LOOP CONTROL CODE:**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{ uint storedData; constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{ uint n = 1; uint sum = 0;**

**while( n < 10){ n++; if(n == 5){ continue; // skip n in sum when it is 5.**

**}**

**sum = sum + n;**

**}**

**return integerToString(sum);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0)**

**{ return "0"; } uint j = \_i;**

**uint len;**

**while (true) { len++; j /= 10;**

**if(j==0){**

**break; //using break statement**

**}**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

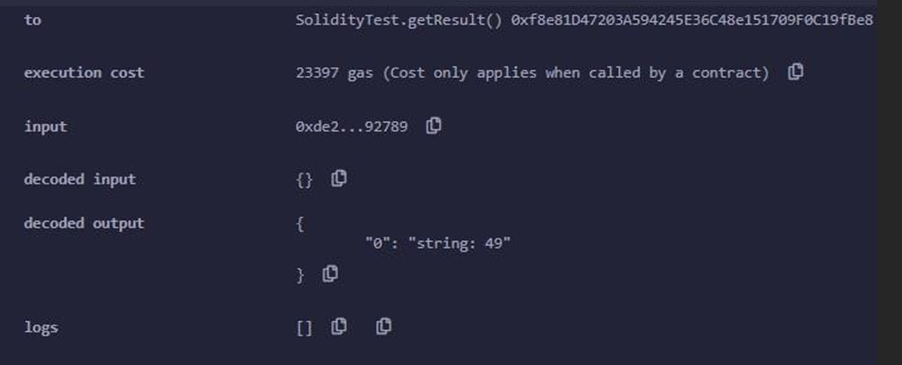
**}**

**return string(bstr);**

**}**

**}**

**OUTPUT:**

****

**DECISION MAKING CODE:**

**If statement codes :**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**uint storedData; constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 1;**

**uint b = 2;**

**uint result = a + b;**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure**

**returns (string memory)**

**{**

**if (\_i == 0)**

**{**

**// if statement**

**return "0";**

**}**

**uint j = \_i;**

**while (j != 0) { len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

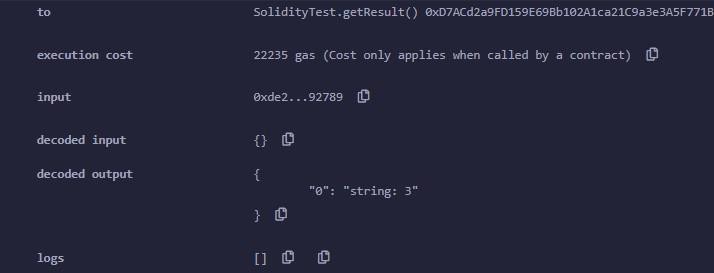
**}**

**return string(bstr); //access local variable**

**}**

**}**

**0UTPUT:**



**IF...ELSE STATEMENT CODE :**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**uint storedData;**

**constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{ uint a = 1; uint b = 2; uint result if( a > b)**

**{ // if else statement**

**result = a;**

**} else { result = b;**

**}**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0) { return "0";**

**}**

**uint j = \_i;**

**uint len;**

**while (j != 0) { len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0) {**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

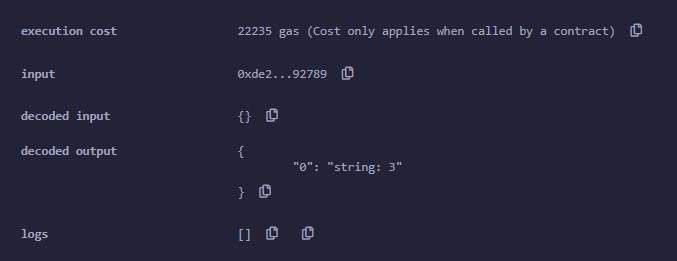
**}**

**return string(bstr);//access local variable**

**}**

**}**

**OUTPUT:**



**IF ELSE IF STATEMENT CODE :**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**uint storedData; // State variable constructor() public**

**{**

**storedData = 10;**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 1;**

**uint b = 2;**

**uint c = 3; uint result**

**if( a > b && a > c) { // if else statement result = a;**

**} else if( b > a && b > c )**

**{**

**result = b;**

**}**

**else**

**{ result = c;**

**}**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0)**

**{**

**return "0";**

**}**

**uint j = \_i;**

**uint len;**

**while (j != 0) { len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len);**

**uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

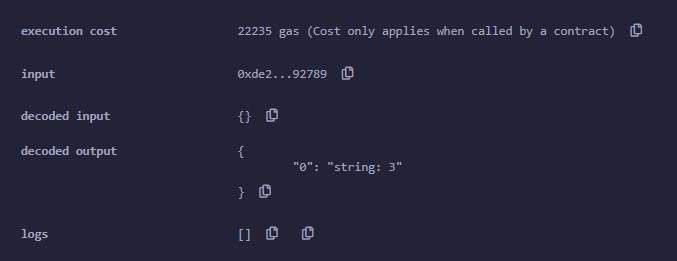
**}**

**return string(bstr);//access local variable**

**}**

**}**

**OUTPUT:**



**STRINGS CODE :**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**constructor()**

**public**

**{**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 1;**

**uint b = 2;**

**uint result = a + b;**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure returns (string memory)**

**{**

**if (\_i == 0) { return "0";**

**}**

**uint j = \_i;**

**uint len;**

**while (j != 0) { len++; j /= 10;**

**}**

**bytes memory bstr = new bytes(len); uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**\_i /= 10;**

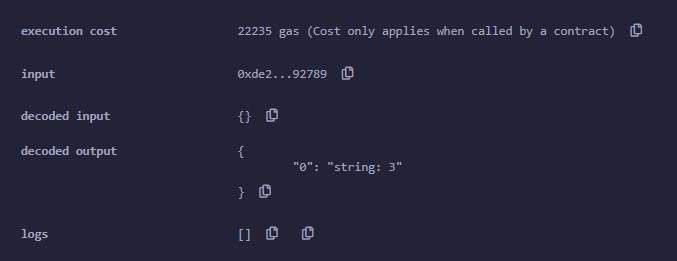
**}**

**return string(bstr);**

**}**

**}**

**OUTPUT:**



**ENUMS CODE :**

**pragma solidity ^0.5.0;**

**contract test**

**{**

**enum FreshJuiceSize**

**{**

**SMALL, MEDIUM, LARGE**

**}**

**FreshJuiceSize choice;**

**FreshJuiceSize constant defaultChoice = FreshJuiceSize.MEDIUM;**

**function setLarge() public**

**{**

**choice = FreshJuiceSize.LARGE;**

**}**

**function getChoice()**

**public view returns (FreshJuiceSize)**

**{**

**return choice;**

**}**

**function getDefaultChoice()**

**public pure returns (uint)**

**{**

**return uint(defaultChoice);**

**}**

**}**

**OUTPUT:**

****

****

**STRUCTS CODE:**

**pragma solidity ^0.5.0;**

**contract test { struct Book**

**{**

**string title;**

**string author; uint book\_id;**

**}**

**Book book;**

**function setBook() public**

**{**

**book = Book('Learn Java', 'TP', 1);**

**}**

**function getBookId() public view returns (uint)**

**{**

**return book.book\_id;**

**}**

**}**

**OUTPUT**



**MAPPINGS CODE:**

**pragma solidity ^0.5.0;**

**contract LedgerBalance**

**{**

**mapping(address => uint)**

**public balances;**

**function updateBalance(uint newBalance) public**

**{**

**balances[msg.sender] = newBalance;**

**}**

**}**

**contract Updater**

**{**

**function updateBalance() public returns (uint)**

**{**

**LedgerBalance ledgerBalance = new LedgerBalance();**

**ledgerBalance.updateBalance(10);**

**return ledgerBalance.balances(address(this));**

**}**

**}**

**OUTPUT:**



**SPECIAL VARIABLES CODE:**

**contract LedgerBalance**

**{**

**mapping(address => uint)**

**public balances;**

**function updateBalance(uint newBalance)**

**public**

**{**

**balances[msg.sender] = newBalance;**

**}**

**}**

**contract Updater**

**{**

**function updateBalance() public returns (uint)**

**{ LedgerBalance ledgerBalance = new LedgerBalance();**

**ledgerBalance.updateBalance(10);**

**return ledgerBalance.balances(address(this));**

**}**

**}**

**OUTPUT:**



1. **Functions, Function Modifiers, View functions, Pure Functions, Fallback Function, Function Overloading, Mathematical functions, Cryptographic functions**

**Function Overloading, Mathematical functions, Cryptographic functions.**

**FUNCTIONS CODE:**

**pragma solidity ^0.5.0;**

**contract SolidityTest**

**{**

**constructor()**

**public**

**{**

**}**

**function getResult() public view returns(string memory)**

**{**

**uint a = 1;**

**uint b = 2;**

**uint result = a**

**+ b;**

**return integerToString(result);**

**}**

**function integerToString(uint \_i) internal pure**

**returns (string memory)**

**{**

**if (\_i == 0)**

**{**

**return “0”;**

**}**

**uint j = \_i;**

**uint len;**

**while (j != 0)**

**{**

**len ++;**

**j /= 10;**

**}**

**bytes memory bstr = new bytes(len);**

**uint k = len - 1;**

**while (\_i != 0)**

**{**

**bstr[k--] = byte(uint8(48 + \_i % 10));**

**i /=10;**

**}**

**return string(bstr);//access local variable**

**}**

**}**

**OUTPUT:**

****

**FALLBACK FUNCTION CODE:**

**pragma solidity ^0.5.0;**

**contract Test**

**{**

**uint public x ;**

**function() external { x = 1; }**

**}**

**contract Sink**

**{**

**function() external payable { }**

**}**

**contract Caller**

**{**

**(bool success,) = address(test).call(abi.encodeWithSignature("nonExistingFunction()")); require(success);**

**// test.x is now 1**

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

**// Sending ether to Test contract,**

**// the transfer will fail, i.e. this returns false here. return (testPayable.send(2 ether));**

**}**

**function callSink(Sink sink) public returns (bool) { address payable sinkPayable = address(sink); return (sinkPayable.send(2 ether));**

**}**

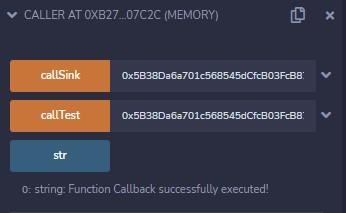
**string public str = "Function Callback successfully executed!";**

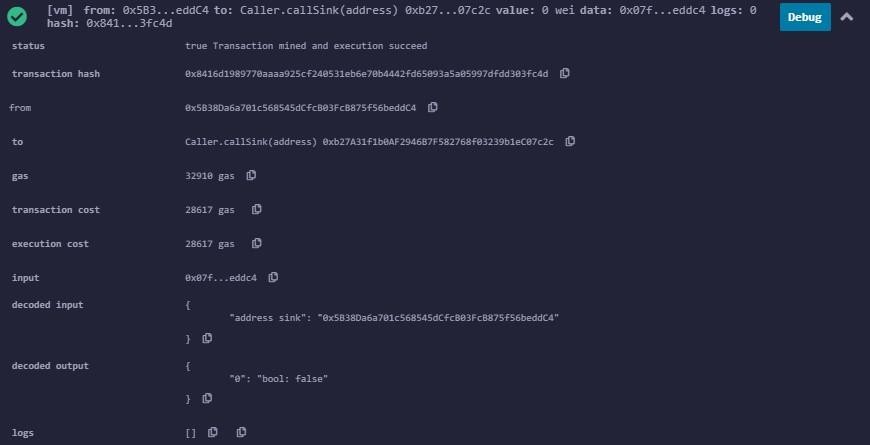
**}**

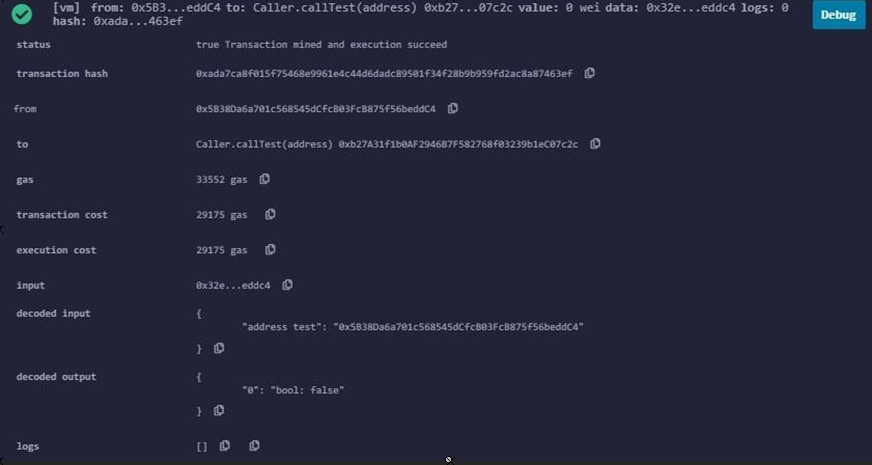
**function callTest(Test test) public returns (bool)**

**}**

**OUTPUT:**





****

**FUNCTION OVERLOADING CODE:**

**pragma solidity ^0.5.0;**

**contract Test**

**{**

**function getSum(uint a, uint b)**

**public pure returns(uint)**

**{**

**return a + b;**

**}**

**function getSum(uint a, uint b, uint c)**

**public pure returns(uint)**

**{**

**return a + b + c;**

**}**

**function callSumWithTwoArguments()**

**public pure returns(uint)**

**{**

**return getSum(1,2);**

**}**

**function callSumWithThreeArguments()**

**public pure returns(uint)**

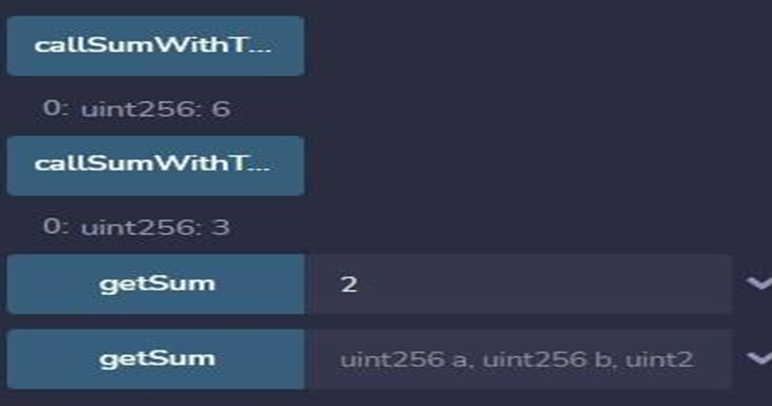
**{**

**return getSum(1,2,3);**

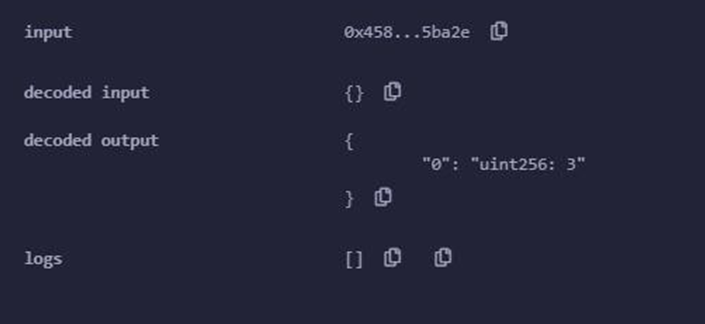
**}**

**}**

**OUTPUT:**

****

****

****

**SOLIDITY - MATHEMATICAL FUNCTIONS CODE**

**pragma solidity ^0.5.0;**

**contract Test**

**{**

**function callAddMod()**

**public pure returns(uint)**

**{**

**return addmod(4, 5, 3);**

**}**

**function callMulMod()**

**public pure returns(uint)**

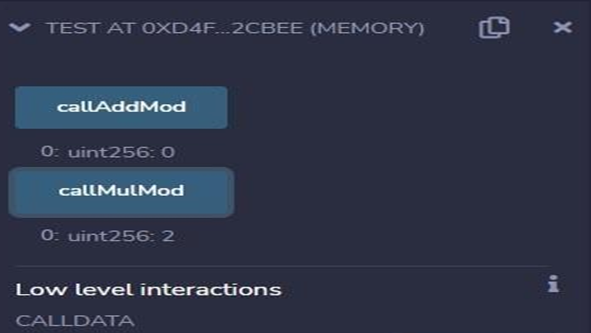
**{**

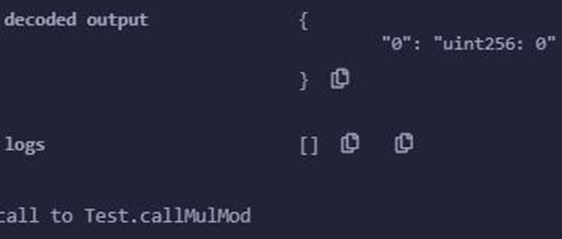
**return mulmod(4, 5, 3);**

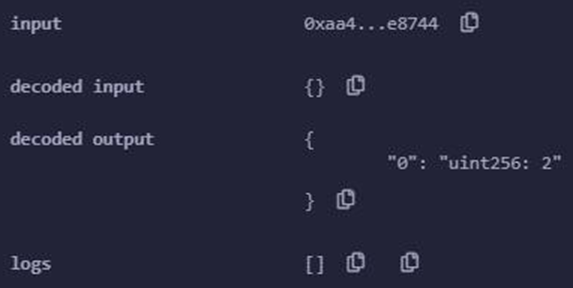
**}**

**}**

**OUTPUT:**

****

****

****

**SOLIDITY - CRYPTOGRAPHIC FUNCTIONS CODE:**

**pragma solidity ^0.5.0;**

**contract Test**

**{**

**function callKeccak256()**

**public pure returns(bytes32 result)**

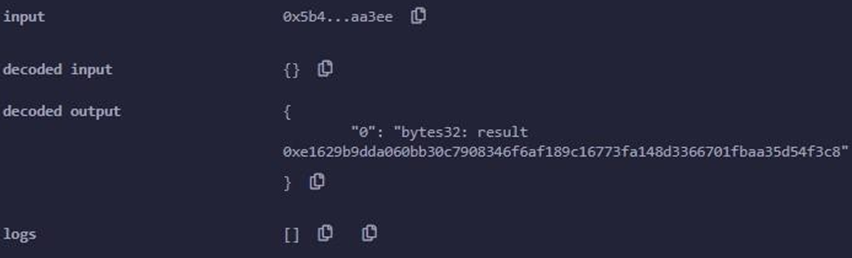
**{**

**return keccak256("ABC");**

**}**

**}**

**OUTPUT:**

****

# PRACTICAL 4

**AIM**

**Implement and demonstrate the use of the following in Solidity:**

**a. Contracts, Inheritance, Interfaces. Contracts Code:**

**pragma solidity ^0.5.0;**

**contract C**

**{**

**//private state variable uint private data;**

**//public state variable uint public info;**

**//constructor constructor() public { info = 10;**

**}**

**//private function**

**function increment(uint a) private pure returns(uint)**

**{**

**return a + 1;**

**}**

**//public function**

**function updateData(uint a)**

**public**

**{**

**data = a;**

**}**

**function getData() public view returns(uint)**

**{**

**return data;**

**}**

**function compute(uint a, uint b) internal pure returns (uint)**

**{**

**return a + b;**

**}**

**}**

**//External Contract contract D**

**{**

**function readData()**

**public returns(uint) { C c = new C();**

**c.updateData(7); return c.getData();**

**}**

**}**

**//Derived Contract contract E is C**

**{**

**uint private result; C private c;**

**constructor() public { c = new C();**

**function getComputedResult()**

**public**

**{**

**result = compute(3, 5);**

**}**

**function getResult()**

**public view returns(uint)**

**{**

**return result;**

**}**

**function getData()**

**public view returns(uint)**

**{**

**return c.info();**

**}**

**}**

****

**INHERITANCE CODE:**

**pragma solidity ^0.5.0;**

**contract C**

**{**

**//private state variable uint private data;**

**//public state variable uint public info;**

**//constructor constructor() public { info = 10;**

**}**

**//private function**

**function increment(uint a)**

**private pure returns(uint)**

**{**

**return a + 1;**

**}**

**//public function**

**function updateData(uint a)**

**public**

**{**

**data = a;**

**}**

**function getData()**

**public view returns(uint)**

**{**

**return data;**

**}**

**function compute(uint a, uint b)**

**internal pure returns (uint) { return a + b;**

**}**

**//Derived Contract contract E is C**

**{**

**uint private result;**

**C private c; constructor()**

**public**

**{**

**c = new C();**

**}**

**function getComputedResult()**

**public**

**{**

**result = compute(3, 5);**

**}**

**function getResult()**

**public view**

**returns(uint)**

**{**

**return result;**

**}**

**function getData()**

**public view returns(uint)**

**{**

**return c.info();**

**}**

**}**

****

**INTERFACES CODE:**

**pragma solidity ^0.5.0;**

**interface Calculator**

**{**

**function getResult()**

**external view returns(uint);**

**}**

**contract Test is Calculator**

**{**

**constructor()**

**public {}**

**function getResult()**

**external view returns(uint)**

**{**

**uint a = 1;**

**uint b = 2;**

**uint result = a + b;**

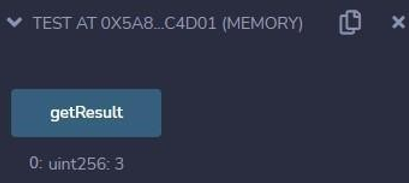
**return result;**

**}**

**}**

**OUTPUT**

****

****

**B. Libraries, Assembly, Error handling.**

**LIBRARIES CODE:**

**pragma solidity ^0.5.0;**

**library Sum**

**{**

**function sumUsingInlineAssembly(uint[] memory \_data)**

**public pure returns (uint o\_sum)**

**{**

**for (uint i = 0; i < \_data.length; ++i)**

**{**

**assembly**

**{**

**o\_sum := add(o\_sum, mload(add(add(\_data, 0x20), mul(i, 0x20))))**

**}**

**}**

**}**

**}**

**contract Test { uint[] data;**

**constructor() public { data.push(1); data.push(2);**

**data.push(3); data.push(4); data.push(5);**

**}**

**function sum()**

**external view returns(uint)**

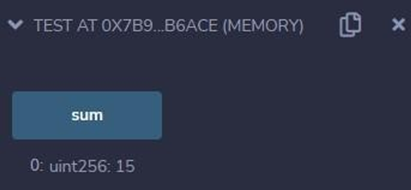
**{**

**return Sum.sumUsingInlineAssembly(data);**

**}**

**}**

**OUTPUT:**

****

**ASSEMBLY CODE:**

**pragma solidity ^0.5.0;**

**library Sum**

**{**

**function sumUsingInlineAssembly(uint[] memory \_data)**

**public pure returns (uint o\_sum)**

**{**

**for (uint i = 0; i < \_data.length; ++i)**

**{**

**assembly**

**{**

**o\_sum := add(o\_sum, mload(add(add(\_data, 0x20), mul(i, 0x20))))**

**}**

**}**

**}**

**}**

**contract Test { uint[] data;**

**constructor() public { 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);**

**}**

**}**

# PRACTICAL 5

**AIM: Create your own blockchain and demonstrate its use.**

**CODE:**

**class Block(object):**

**def init (self, index, proof\_number, previous\_hash, data, timestamp=None):**

**self.index = index self.proof\_number = proof\_number self.previous\_hash = previous\_hash self.data = data self.timestamp = timestamp or time.time()**

**@property def compute\_hash(self):**

**string\_block = "{}{}{}{}{}".format(self.index, self.proof\_number, self.previous\_hash, self.data, self.timestamp)**

**return hashlib.sha256(string\_block.encode()).hexdigest() def repr (self):**

**return "{} - {} - {} - {} - {}".format(self.index, self.proof\_number, self.previous\_hash, self.data, self.timestamp)**

**class BlockChain(object) def init**

**(self):**

**self.chain = [] self.current\_data = []**

**self.nodes = set() self.build\_genesis()**

**def build\_genesis(self):**

**self.build\_block(proof\_number=0, previous\_hash=0)**

**def build\_block(self, proof\_number, previous\_hash):**

**block = Block( index=len(self.chain), proof\_number=proof\_number, previous\_hash=previous\_hash, data=self.current\_data)**

**self.current\_data = []**

**self.chain.append(block) return block @staticmethod**

**def confirm\_validity(block, previous\_block):**

**if previous\_block.index + 1 != block.index:**

**return False**

**elif previous\_block.compute\_hash != block.previous\_hash:**

**return False**

**elif block.timestamp <= previous\_block.timestamp:**

**return False**

**return True**

**def get\_data(self, sender, receiver, amount):**

**self.current\_data.append**

**({'sender': sender, 'receiver': receiver, 'amount': amount})**

**return True @staticmethod def proof\_of\_work(last\_proof):**

**pass @property def latest\_block(self):**

**return self.chain[-1] def chain\_validity(self):**

**pass def block\_mining(self, details\_miner):**

**self.get\_data( sender="0", #it implies that this node has created a new block receiver=details\_miner, quantity=1, #creating a new block (or identifying the proof number) is awared with 1)**

**last\_block = self.latest\_block last\_proof\_number = last\_block.proof\_number proof\_number =**

**self.proof\_of\_work(last\_proof\_number) last\_hash =**

**last\_block.compute\_hash block = self.build\_block(proof\_number, last\_hash) return vars(block) def create\_node(self, address): self.nodes.add(address) return True @staticmethod**

**def get\_block\_object(block\_data): return Block( block\_data['index'],**

**block\_data['proof\_number'], block\_data['previous\_hash'], block\_data['data'], timestamp=block\_data['timestamp'**

**]**

**)**

**blockchain = BlockChain()**

**print("GET READY MINING ABOUT TO START")**

**print(blockchain.chain)**

**last\_block = blockchain.latest\_block last\_proof\_number = last\_block.proof\_number**

**proof\_number = blockchain.proof\_of\_work(last\_proof\_number) blockchain.get\_data( sender="0", #this means that this node has constructed another block receiver="sharik", amount=1, #building a new block (or figuring out the proof number) is awarded with 1**

**)**

**last\_hash = last\_block.compute\_hash**

**block = blockchain.build\_block(proof\_number, last\_hash)**

**print("WOW, MINING HAS BEEN SUCCESSFUL!")**

**print(blockchain.chain)**

**OUTPUT:**

