Practical 1 : Demonstrate the use of different types of variables in solidity

Code :

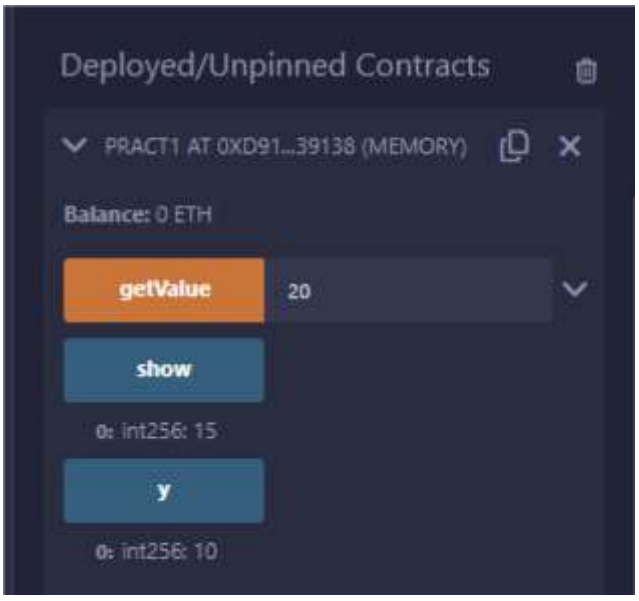
```
pragma solidity ^0.5.0; contract
Pract1{ int x=15; //state var int
public y=10;//global function
getValue(int z) public{ y=y+z;

} function show() public view returns
(int)

{ return
x;

}
}
```

Output :



Practical 2 : Write a solidity program to demonstrate relational operators.

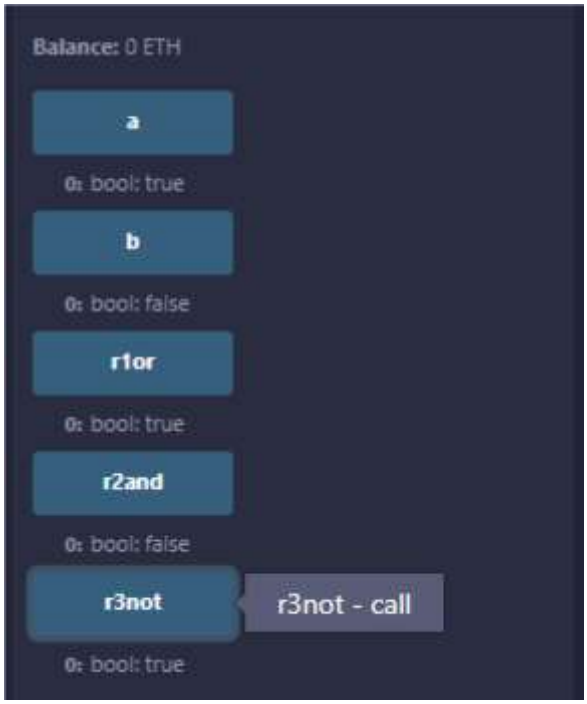
Code :

```
pragma solidity ^0.5.0;
contract Pract2{ bool
public a=true; bool
public b=false; bool
public r1or=a||b;

bool public r2and=a&&b; bool
public r3not=!b;

}
```

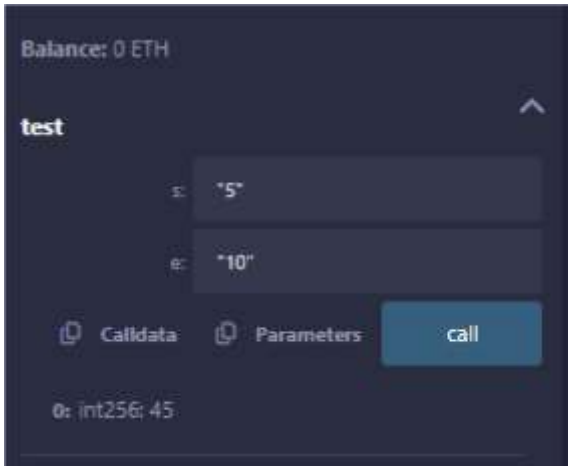
Output :



Practical 3 : Write a Solidity program to print sum of 10 numbers using for loop Code :

```
pragma solidity ^0.5.0; contract
Pract3{
function test(int s, int e) public view returns(int)
{ int
i;
int sum=0; for(i=s;i<=e;i++)
{
sum+=i;  //sum=sum+i;
} return
sum;
}
}
```

Output :

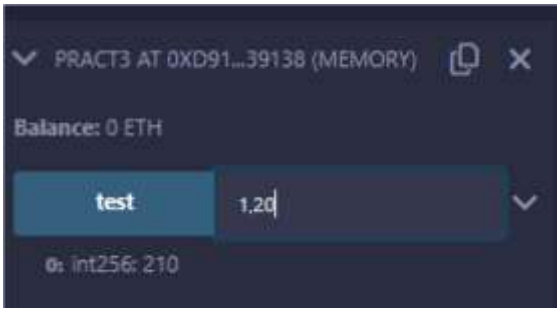


Practical 4 : Write a Solidity program to print sum of 10 numbers using while loop.

Code :

```
pragma solidity ^0.5.0; contract
Pract3{
function test(int s, int e) public view returns(int)
{ int
i;
int sum=0;
i=s; while(i<=e)
{
sum+=i;  //sum=sum+i;
i++;
} return
sum;
}
}
```

Output :

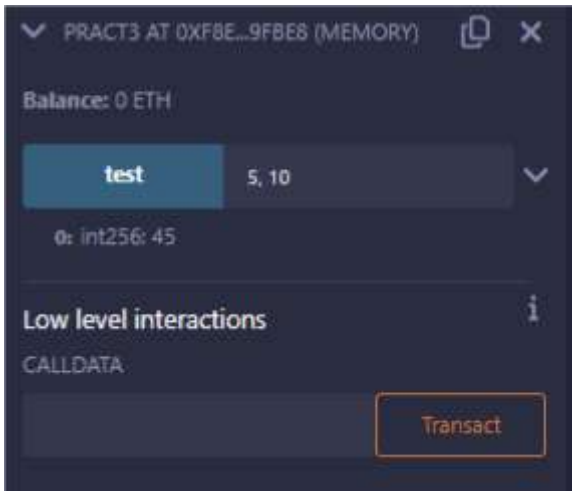


Practical 5 : Write a Solidity program to print sum of 10 numbers using do while.

Code :

```
pragma solidity ^0.5.0; contract
Pract3{
function test(int s, int e) public view returns(int)
{ int
i;
int sum=0;
i=s; do
{
sum+=i;  //sum=sum+i;
i++;
}while(i<=e); return
sum;
}
}
```

Output :

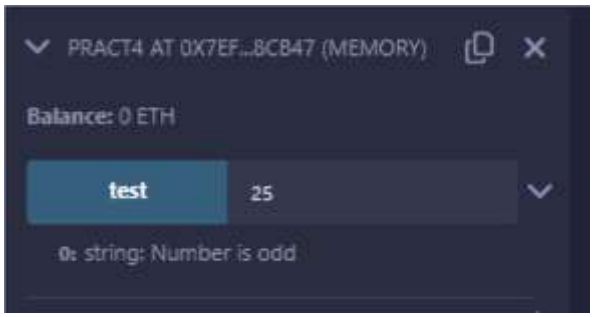


Practical 6 : Write a Solidity program to check if number is even or odd.

Code :

```
pragma solidity ^0.5.0; contract
Pract4{
function test(int x) public view returns(string memory)
{
if(x%2==0)
return "Number is even"; else
return "Number is odd";
}
}
```

Output :

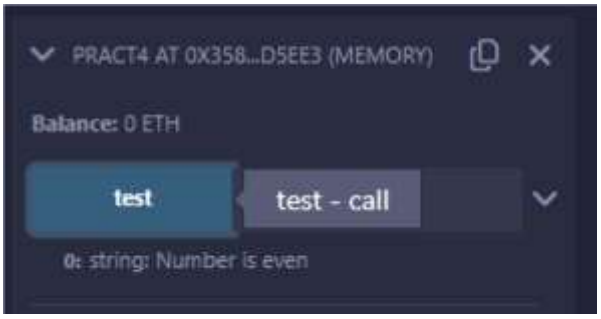


Practical 7 : Write a Solidity program to use string.

Code :

```
pragma solidity ^0.5.0; contract
Pract4{
function test(int x) public view returns(string memory)
{
if(x%2==0)
return "Number is even"; else
return "Number is odd";
}
}
```

Output :



Practical 8 : Demonstrate the use of array. Also find sum of array.

Code:

```
contract Types {
uint[5] data;
constructor() public
{
   data = [uint(10), 20, 30, 40, 50];
}
function array_example() public view returns (uint,uint) {

return (data[0],data[4]);
}
function array_example2() public view returns (uint [5] memory) {

return data;
}
}
```

Output :

Practical 9 : Write a Solidity program to use enumeration.

Code :

```
pragma solidity ^0.5.0;
contract Types { enum
week_days
{
Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,
Sunday
}
week_days choice; function
set_value() public { choice =
week_days.Thursday;
}
function get_choice() public view returns (week_days) { return
choice;
}
}
```

Output :

Practical 10 : Write a Solidity program to use arithmetic operations

Code :

```
pragma solidity ^0.5.0; // Creating a contract contract SolidityTest { // Initializing variables uint16 public a = 20; uint16 public b = 10; // Initializing a variable

// with sum uint public sum = a + b; // Initializing a variable // with the difference uint public diff = a - b;

// Initializing a variable // with product uint public mul = a * b;

// Initializing a variable // with quotient uint public div = a / b; // Initializing a variable // with modulus uint public mod = a % b;
```

```
// Initializing a variable
// decrement value uint
public dec = --b;
// Initializing a variable //
with increment value
uint public inc = ++a;

}
```

Output :

Practical 11 : Create a book structure, assign values and display the values using Solidity.

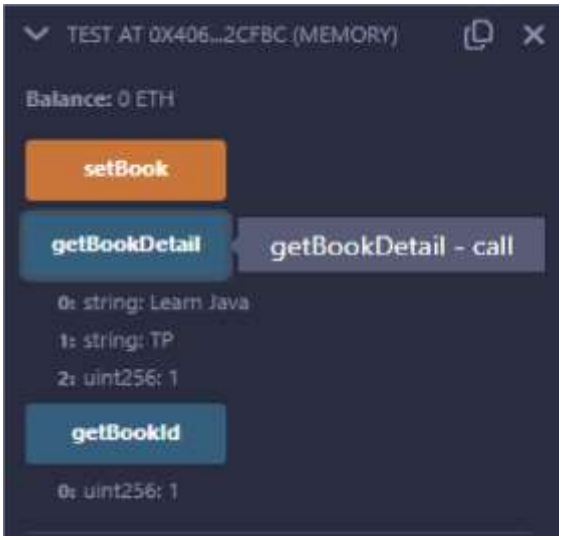
Code :

```
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;
}
function getBookDetail() public view returns (string memory,
string memory,uint) {
return (book.title, book.author, book.book_id); }

}
```

Output :


TEST AT 0X406...2CFBC (MEMORY)
Balance: 0 ETH
setBook
getBookDetail
getBookDetail - call
0: string: Learn Java
1: string: TP
2: uint256: 1
getBookId
0: uint256: 1

Practical 12 : Write a solidity program to create view and pure function.

Code :

```
pragma solidity ^0.5.0; contract
Test { int public x=10; //global
int y=90;//state function f1()
public returns(int){     //read and
update is allowed     x=100;
return x;
}
function f2() public view returns(int){   //
x=100; //erro beacuse x is global/state
  //we can access but we cannot update state or global variable int
view function return x;
}
function f3() public pure returns(int){
    //we cannot access or update state or global variable in pure
function     int z=80; return z;
}

}
```

Output :

Practical 13 : Write a solidity program to implement 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;
}
}
```

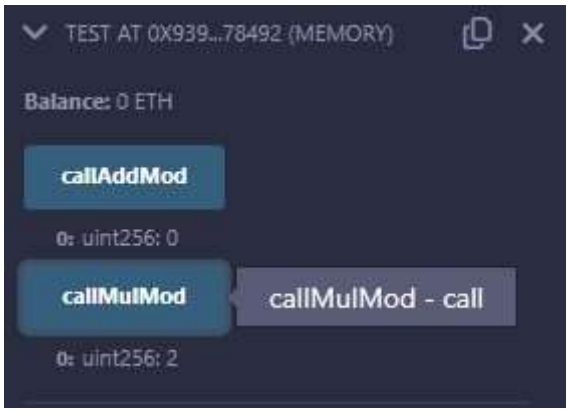
Output :

Practical 14 : Write a solidity program to use mathematical function.

```
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);
}
}
```

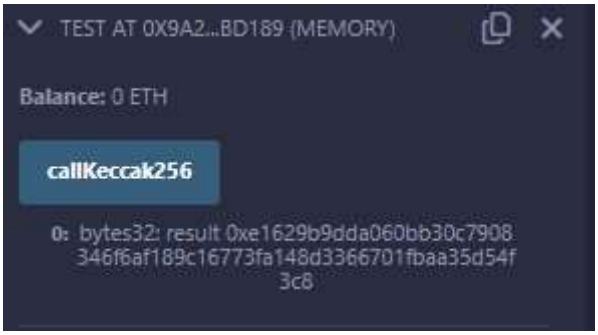
Output :



Practical 15 : Write a solidity program to use cryptographic function.

```
pragma solidity ^0.5.0; contract
Test {
function callKeccak256() public pure returns(bytes32 result){
return keccak256("ABC");
}
}
```

Output :



Practical 16 : Write a Python program to create a simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.

Code :

```
!pip install crypto

!pip install pycrypto !pip

install pycryptodome

import hashlib import

random import string

import json import

binascii import numpy as

np import pandas as pd


import logging import

datetime import

collections

from Crypto.PublicKey import RSA from

Crypto import Random from

Crypto.Cipher import PKCS1_v1_5


class Client:    def

__init__(self):

    random = 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')

Dinesh = Client()

print ("sender ",Dinesh.identity) **Output**

:

Practical 17 : Write a Python program to create a transaction class to send and receive money and test it.

Code :

```
!pip install crypto
!pip install pycrypto !pip
install pycryptodome
import hashlib import
random import binascii
import datetime import
collections


from Crypto.PublicKey import RSA from
Crypto import Random from Crypto.Cipher
import PKCS1_v1_5 from collections
import OrderedDict import Crypto import
Crypto.Random from Crypto.Hash import
SHA from Crypto.Signature import
PKCS1_v1_5 class Client:     def
__init__(self):

     random = 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()


t1 = Transaction(
Dinesh,

  Ramesh.identity,
  15.0
)
t1.sign_transaction()
display_transaction (t1)
```

Output

:


sender: 30819f300d06092a864886f70d010101050003818d00308189028181
recipient: 30819f300d06092a864886f70d010101050003818d00308189028181
value: 15.0
time: 2024-04-03 07:47:03.73241

Practical 18 : Write a Python program to create a mining function and test it.

Code :

```
import hashlib


def sha256(message):
    return hashlib.sha256(message.encode('ascii')).hexdigest()


def mine(message, difficulty=1):
  assert difficulty >= 1
#if(difficulty <1):
  #       return    #'1'*3=>
'111'    prefix = '1' *
difficulty
print("prefix",prefix)
for i in range(100000):
    digest = sha256(str(hash(message)) + str(i))
print("testing=>"+digest)       if digest.startswith(prefix):
print ("after " + str(i) + " iterations found nonce: "+ digest)
return i        #i= nonce value


mine ("test message",3) Output
:
```

```
testing=>39b3babda85a1e258fe0999b0e99ea26034e95b8bd687f4e9c01db92593ecb7f
testing=>01134d677768d116f4fb13e43d5adc93d405b8d90871cccc0c61e959b38d767a
testing=>46c235e4b79ae3c3a09f63104f60f5d8c4757ac04853b06c4638eeb74596ff89
testing=>f39d1fbf874a8e5c4fb55a3a38d20afe29dd17686513b673d7f20825ad3b0c53
testing=>e5eefac14a68f71d571b998af1a06379f77815c630ff81f029220488d4be396e
testing=>ffc239ff843b3dcd787b123475204a8919142985df7036e1651ef4f4da0b8cc6
testing=>aac43fee40a9369bb6fb09aa2441a5f81afa06ecc6c0ef432be95e6aaf04d5a0
testing=>ae194785098b94682bc171da6f19fc64e041eca99f0ceae085fee1e369aa1dbf
testing=>214b2c6e1bac96abfcaa826eb78c3ec9d861ee5137e9174690db2e8f13636115
testing=>0220868fb2955026abf85e1659c1fd35ac6523bcb17e51ccc5755dabde80c7aa
testing=>1d6377a63ef2bfef06e8442d331f3e40f7c00878ca03001971c01470f5e4cadc
testing=>886094746385549df9956b2198eec92b5b13d157b4e6c4c70de9c6b930b00ff8
testing=>972ecaf5f4f74db99d46a122680b093b90ff9a81cc68a7b9d24ca6c8e1e6b103
testing=>d4999afe937f9e1b2048885f2a04e17468c558d0d6f126fc2cf239c1fcfe3b05
testing=>cf706857677496652edeb7d660ece7c49d218ca1b0672941d536a346b99f2fb6
testing=>abead7f6a8bcf37a1aebe22003443e7d4e5d81a6a874cf9fac5606ae16cf4d68
testing=>c2da8700793209d2c441c42b04ebd0a34d5907ac4f2944f80cf71a85922413d6
testing=>c2845b376bdd42a65b7a9c2222c446b66c640bd2c59f74ed976703d914c2c3b4
testing=>a4bed4a4a5399fa2a4b5686b74702780ad4800c1ed01600d628d2b3e00e8c454
testing=>2f3a19e7789b6a7aa083700f4efe7cc6de3ee37c5fb0ea269f5bf736434b216d
testing=>ca7b104814380756b9820de970de27d1da1d27cdf0bf79f8d17721c173c4a48f
testing=>a697ef443137d3e7ba39c44c0546ce3293377300e117a831faf90d2bc1959a98
testing=>111feba0c02824299eceaf7983bff588595818b181ec4b40700ec064fc10f1de
after 10023 iterations found nonce: 111feba0c02824299eceaf7983bff588595818b181ec4b40700ec064fc10f1de
10023
```

Practical 19 : Demonstrate the use of Bitcoin Core API.

Code :

```
#pip install bitcoinlib

from bitcoinlib.wallets import Wallet

w = Wallet.create('Wallet3') key1 =

w.get_key() print(key1.address)

w.scan()

print(w.info())
```

Output :