INDEX

6	**Implement and demonstrate the user of the following in Solidity:** i. Functions ii. View Functions iii. Pure Functions iv. Fallback Functions v. Function Overloading vi. Mathematical Functions vii. Cryptographic Functions			
7	**Implement and demonstrate the user of the following in Solidity:** i. Contracts ii. Inheritance iii. Constructors iv. Abstract Class v. Interfaces			
8	**Implement and demonstrate the user of the following in Solidity:** i. Libraries ii. Assembly iii. Events iv. Error Handling			

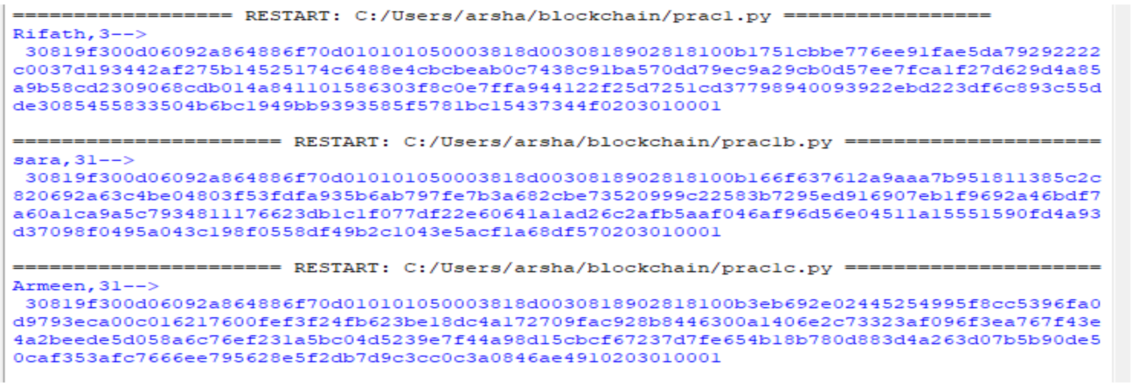
PRACTICAL 1

1. Create a simple client class that generates the private and public keys by using the built-in python RSA algorithm and test it

Code:

```
#import random
from Crypto.PublicKey import RSA
from Crypto import Random
import binascii
from Crypto.Cipher import PKCS1_v1_5

class Client:
   def __init__(self):
      random=Random.new().read
      self._private_key=RSA.generate(1024,random) #1024->key size
      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')

Rifath=Client()
print('Rifath,3--> \n',Rifath.identity)
```

Output:



```
================== RESTART: C:/Users/arsha/blockchain/prac1.py ==================
Rifath,3-->
 30819f300d06092a864886f70d010101050003818d0030818902818100b1751cbbe776ee91fae5da79292222
c0037d193442af275b14525174c6488e4cbcbeab0c7438c91ba570dd79ec9a29cb0d57ee7fca1f27d629d4a85
a9b58cd2309068cdb014a841101586303f8c0e7ffa944122f25d7251cd37798940093922ebd223df6c893c55d
de3085455833504b6bc1949bb9393585f5781bc15437344f0203010001

====================== RESTART: C:/Users/arsha/blockchain/prac1b.py ======================
sara,31-->
 30819f300d06092a864886f70d010101050003818d0030818902818100b166f637612a9aaa7b951811385c2c
820692a63c4be04803f53fdfa935b6ab797fe7b3a682cbe73520999c22583b7295ed916907eb1f9692a46bdf7
a60a1ca9a5c7934811176623db1c1f077df22e60641a1ad26c2afb5aaf046af96d56e04511a15551590fd4a93
d37098f0495a043c198f0558df49b2c1043e5acf1a68df570203010001

====================== RESTART: C:/Users/arsha/blockchain/prac1c.py ======================
Armeen,31-->
 30819f300d06092a864886f70d010101050003818d0030818902818100b3eb692e02445254995f8cc5396fa0
d9793eca00c016217600fef3f24fb623be18dc4a172709fac928b8446300a1406e2c73323af096f3ea767f43e
4a2beede5d058a6c76ef231a5bc04d5239e7f44a98d15cbcf67237d7fe654b18b780d883d4a263d07b5b90de5
0caf353afc7666ee795628e5f2db7d9c3cc0c3a0846ae4910203010001
```

2.
3.
4.
5.
6.
7.
8.
9.
10.

A transaction class to send and receive money and test it

Code:

```
#import random
from Crypto.PublicKey import RSA
from Crypto import Random
import binascii
from Crypto.Cipher import PKCS1_v1_5
from Crypto.Hash import SHA
import datetime
import collections
from Crypto.Signature import PKCS1_v1_5
from collections import OrderedDict

class Client:
    def __init__(self):
        random=Random.new().read
        self._private_key=RSA.generate(1024,random) #1024->key size
        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,receiver,value):
        self.sender=sender
        self.receiver=receiver
        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,
            "receiver":self.receiver,
            "value":self.value,
            "time":self.time
        })
    def sign_tran(self):
        private_key=self.sender._private_key
        signer=PKCS1_v1_5.new(private_key)
        h=SHA.new(str(self.to_dict).encode('utf-8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')

def display_tran(transaction):
    dict=transaction.to_dict()
    print('\nsender,Rifath--> \n'+dict['sender'])
    print('\nreceiver,Sara--> \n'+dict['receiver'])
    print('\nvalue--> \n'+str(dict['value']))
    print('\ntime--> \n'+str(dict['time']))

transactions=[]

Rifath=Client()
Sara= Client()

t1=Transaction(
Rifath,
Sara.identity,
15)

t1.sign_tran()
display_tran(t1)
```

PRACTICAL 2

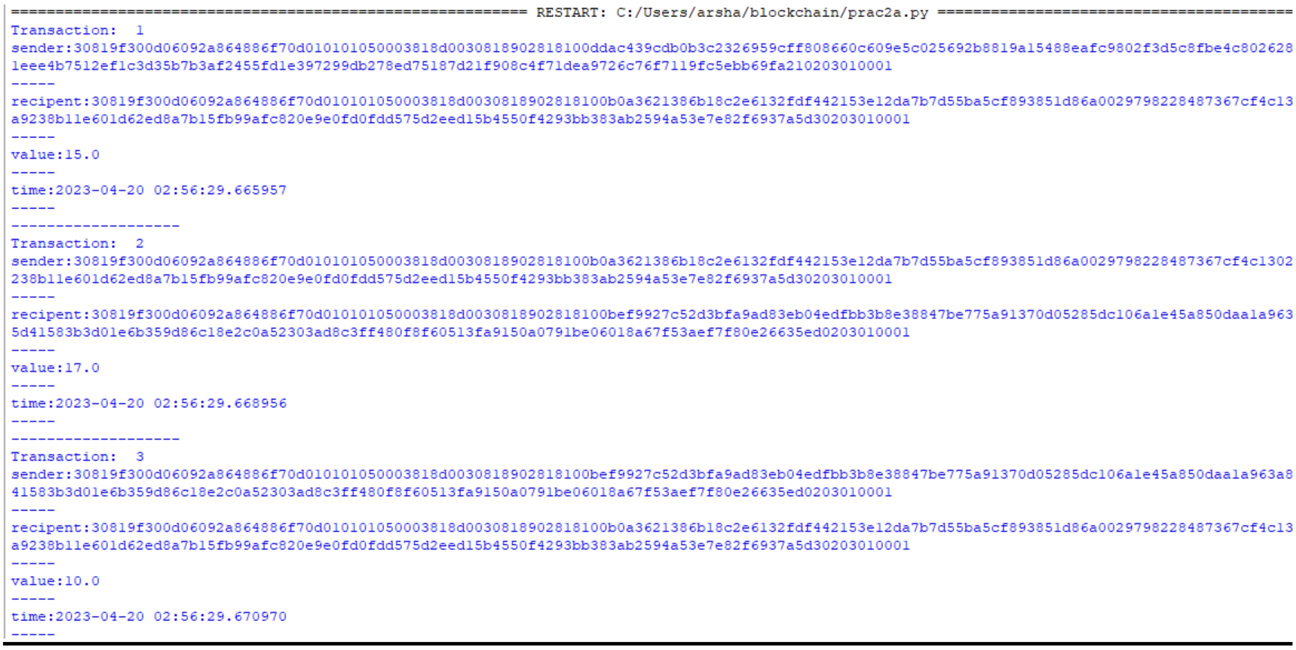
1. Create multiple transactions and display them

Code:

```
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
import datetime
import binascii
from collections import OrderedDict
import collections
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, recipent, value):
        self.sender = sender
        self.recipent = recipent
        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,
            'recipent': self.recipent,
            'value': self.value,
            'time': self.time
        })
    def sign_tran(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("recipent:" + dict['recipent'])
    print('-----')
    print("value:" + str(dict['value']))
    print('-----')
    print("time:" + str(dict['time']))
    print('-----')
transactions = []
Rifath = Client()
Armeen = Client()
Sara = Client()
```

```
t1 = Transaction(
   Rifath,
   Armeen.identity,
   15.0
)
t1.sign_tran()
transactions.append(t1)
t2 = Transaction(
   Armeen,
   Sara.identity,
   17.0
)
t2.sign_tran()
transactions.append(t2)
t3 = Transaction(
   Sara,
   Armeen.identity,
   10.0
)
t3.sign_tran()
transactions.append(t3)
tn = 1
for t in transactions:
   print("Transaction: ", tn)
   display_transaction(t)
   tn = tn + 1
   print('--------------------')
```

Output:



```
============================================================ RESTART: C:/Users/arsha/blockchain/prac2a.py ==========================================
Transaction:  1
sender:30819f300d06092a864886f70d010101050003818d0030818902818100ddac439cdb0b3c2326959cff808660c609e5c025692b8819a15488eafc9802f3d5c8fbe4c802628
1eee4b7512ef1c3d35b7b3af2455fd1e397299db278ed75187d21f908c4f71dea9726c76f7119fc5ebb69fa210203010001
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100b0a3621386b18c2e6132fdf442153e12da7b7d55ba5cf893851d86a0029798228487367cf4c13
a9238b11e601d62ed8a7b15fb99afc820e9e0fd0fdd575d2eed15b4550f4293bb383ab2594a53e7e82f6937a5d30203010001
-----
value:15.0
-----
time:2023-04-20 02:56:29.665957
-----
--------------------
Transaction:  2
sender:30819f300d06092a864886f70d010101050003818d0030818902818100b0a3621386b18c2e6132fdf442153e12da7b7d55ba5cf893851d86a0029798228487367cf4c1302
238b11e601d62ed8a7b15fb99afc820e9e0fd0fdd575d2eed15b4550f4293bb383ab2594a53e7e82f6937a5d30203010001
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100bef9927c52d3bfa9ad83eb04edfbb3b8e38847be775a91370d05285dc106a1e45a850daa1a963
5d41583b3d01e6b359d86c18e2c0a52303ad8c3ff480f8f60513fa9150a0791be06018a67f53aef7f80e26635ed0203010001
-----
value:17.0
-----
time:2023-04-20 02:56:29.668956
-----
--------------------
Transaction:  3
sender:30819f300d06092a864886f70d010101050003818d0030818902818100bef9927c52d3bfa9ad83eb04edfbb3b8e38847be775a91370d05285dc106a1e45a850daa1a963a8
41583b3d01e6b359d86c18e2c0a52303ad8c3ff480f8f60513fa9150a0791be06018a67f53aef7f80e26635ed0203010001
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100b0a3621386b18c2e6132fdf442153e12da7b7d55ba5cf893851d86a0029798228487367cf4c13
a9238b11e601d62ed8a7b15fb99afc820e9e0fd0fdd575d2eed15b4550f4293bb383ab2594a53e7e82f6937a5d30203010001
-----
value:10.0
-----
time:2023-04-20 02:56:29.670970
-----
```

Create a block chain a Genesis block and execute it.

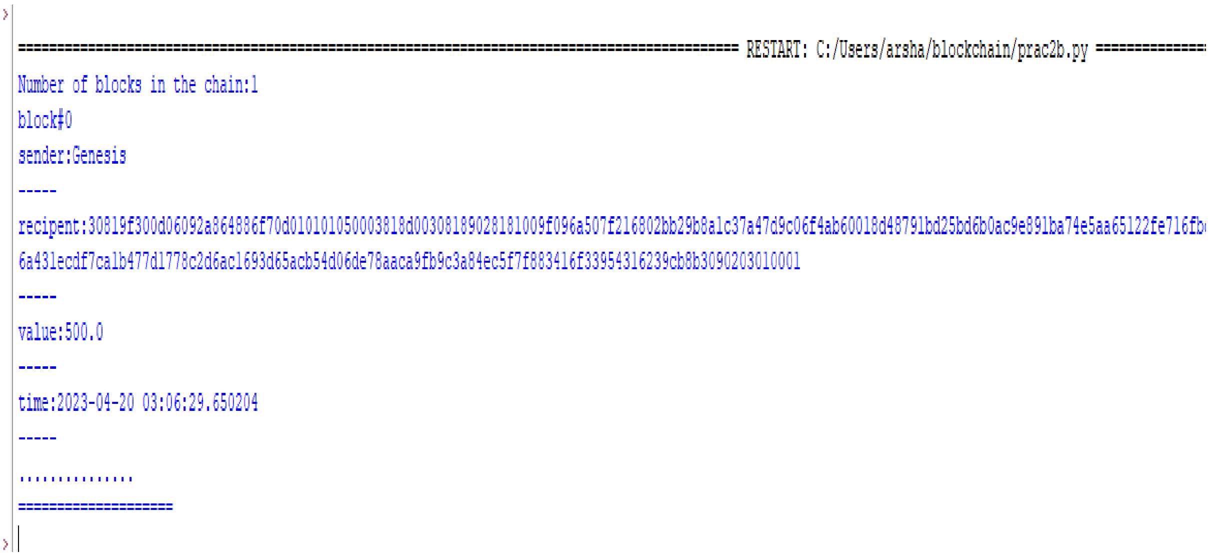
Noance: a randomly generated number (unique) used once in cryptography transaction

Code:

```
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
import datetime
import binascii
from collections import OrderedDict
import collections
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, recipent, value):
        self.sender = sender
        self.recipent = recipent
        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,
            'recipent': self.recipent,
            'value': self.value,
            'time': self.time
        })

    def sign_tran(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("recipent:" + dict['recipent'])
    print('-----')
    print("value:" + str(dict['value']))
    print('-----')
    print("time:" + str(dict['time']))
    print('-----')
```

```
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_transaction:
            display_transaction(transaction)
            print("...............")
            print("====================")

class Block:
    def __init__(self):
        self.verified_transaction=[]
        self.previous_block_hash=""
        self.Nonce=""
Rifath = Client()
t0=Transaction(
    "Genesis",
    Rifath.identity,
    500.0
)
block0=Block()
block0.previous_block_hash=None
Nonce=None
block0.verified_transaction.append(t0)
digest=hash(block0)
last_block_hash = digest
TPCoins=[]
TPCoins.append(block0)
dump_blockchain(TPCoins)
```

Output:



```
======================================================================================== RESTART: C:/Users/arsha/blockchain/prac2b.py ===============
Number of blocks in the chain:1
block#0
sender:Genesis
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100 9f096a507f216802bb29b8a1c37a47d9c06f4ab60018d48791bd25bd6b0ac9e891ba74e5aa65122fe716fb
6a431ecdf7ca1b477d1778c2d6ac1693d65acb54d06de78aaca9fb9c3a84ec5f7f883416f33954316239cb8b3090203010001
-----
value:500.0
-----
time:2023-04-20 03:06:29.650204
-----
...............
====================
```

PRACTICAL 3

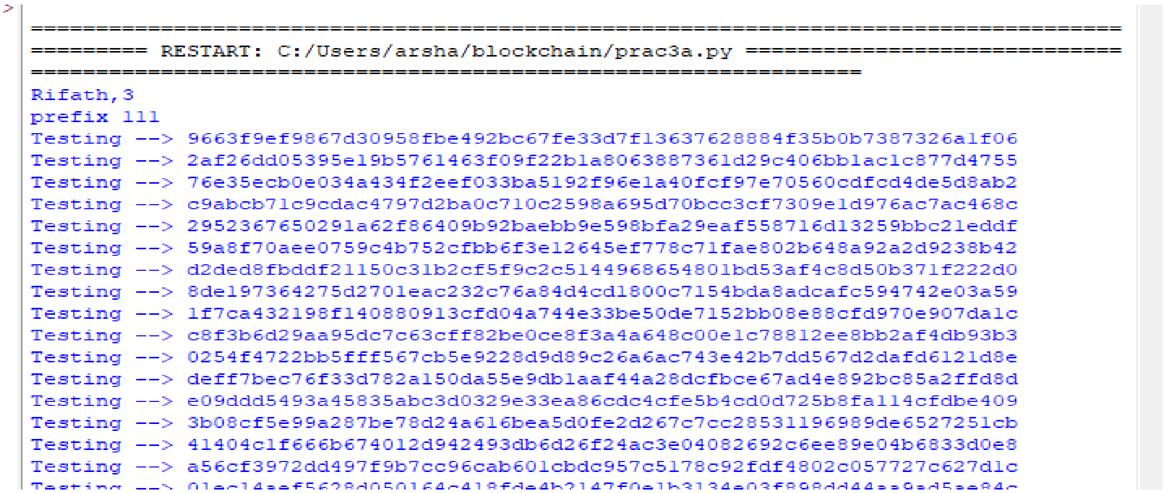
1. Create a mining function and test it.

Miners: verifies the transactions in block chain

Code:

```
print("Rifath,3")
import hashlib

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

def mine(message,difficulty=1):
   assert difficulty>=1 #debugging
   prefix= '1'* difficulty #verify diffficulty
   print ("prefix",prefix)
   for i in range(1000):
      digest = sha256(str(hash(message)) + str(i))
      print("Testing --> " + digest)
      if digest.startswith(prefix):
         print("After" + str(i) + "iterations found nounce" + digest)
         return i
mine("Rifath", 3)
```

Output:



```
>
================================================================================
========= RESTART: C:/Users/arsha/blockchain/prac3a.py ============================
================================================================
Rifath,3
prefix 111
Testing --> 9663f9ef9867d30958fbe492bc67fe33d7f13637628884f35b0b7387326a1f06
Testing --> 2af26dd05395e19b5761463f09f22b1a8063887361d29c406bb1ac1c877d4755
Testing --> 76e35ecb0e034a434f2eef033ba5192f96e1a40fcf97e70560cdfcd4de5d8ab2
Testing --> c9abcb71c9cdac4797d2ba0c710c2598a695d70bcc3cf7309e1d976ac7ac468c
Testing --> 295236765029la62f86409b92baebb9e598bfa29eaf558716d13259bbc21eddf
Testing --> 59a8f70aee0759c4b752cfbb6f3e12645ef778c71fae802b648a92a2d9238b42
Testing --> d2ded8fbddf21150c31b2cf5f9c2c514496865480lbd53af4c8d50b371f222d0
Testing --> 8de197364275d2701eac232c76a84d4cd1800c7154bda8adcafc594742e03a59
Testing --> 1f7ca432198f140880913cfd04a744e33be50de7152bb08e88cfd970e907da1c
Testing --> c8f3b6d29aa95dc7c63cff82be0ce8f3a4a648c00e1c78812ee8bb2af4db93b3
Testing --> 0254f4722bb5fff567cb5e9228d9d89c26a6ac743e42b7dd567d2dafd6121d8e
Testing --> deff7bec76f33d782a150da55e9db1aaf44a28dcfbce67ad4e892bc85a2ffd8d
Testing --> e09ddd5493a45835abc3d0329e33ea86cdc4cfe5b4cd0d725b8fa114cfdbe409
Testing --> 3b08cf5e99a287be78d24a616bea5d0fe2d267c7cc28531196989de6527251cb
Testing --> 41404c1f666b674012d942493db6d26f24ac3e04082692c6ee89e04b6833d0e8
Testing --> a56cf3972dd497f9b7cc96cab601cbdc957c5178c92fdf4802c057727c627d1c
Testing --> 01ec14aef5628d050164c418fde4b2147f0e1b3134e03f898dd44aa9ad5ae84c
```

2. Add block to miner and dump the block chain.

Miners: verifies the transactions in block chain

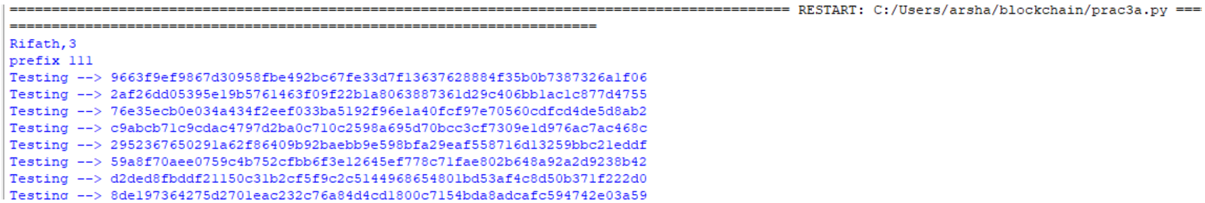
Code:

```
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import PKCS1_v1_5
import datetime
import binascii
from collections import OrderedDict
import collections
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5
import hashlib
```

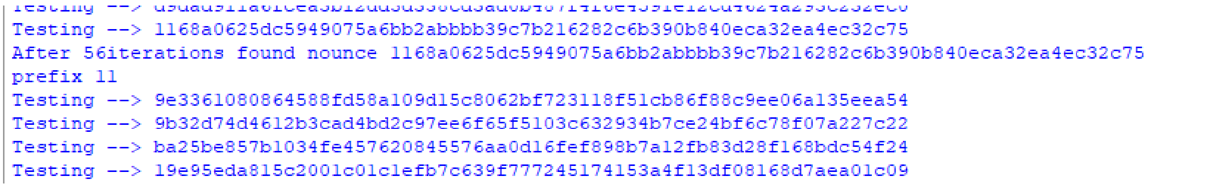
```
print("Rifath,27")
class Client:
   def __init__(self):
      random = Random.new().read
      self._private_key = RSA.generate(1024, random)
      self._public_key = self._private_key.publickey()
      self._signer = PKCS1_v1_5.new(self._private_key)

   @property
   def identity(self):
      return binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')


class Transaction:
   def __init__(self, sender, recipent, value):
      self.sender = sender
      self.recipent = recipent
      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,
         'recipent': self.recipent,
         'value': self.value,
         'time': self.time
      })

   def sign_tran(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("recipent:" + dict['recipent'])
   print('-----')
   print("value:" + str(dict['value']))
   print('-----')
   print("time:" + str(dict['time']))
   print('-----')

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_transaction:
         display_transaction(transaction)
         print("...............")
         print("===================")

class Block:
    def __init__(self):
        self.verified_transaction=[]
        self.previous_block_hash=""
        self.Nonce=""

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

def mine(message,difficulty=1):
    assert difficulty>=1 #debugging
    prefix= '1'* difficulty #verify diffficulty
    print ("prefix",prefix)
    for i in range(1000):
        digest = sha256(str(hash(message)) + str(i))
        print("Testing --> " + digest)
        if digest.startswith(prefix):
            print("After " + str(i) + "iterations found nounce " + digest)
            return i
mine("Rifath", 3)

transactions = []

Rifath = Client()
Armeen = Client()
Sara = Client()

t0=Transaction(
    "Genesis",
    Rifath.identity,
    500.0
)

t1 = Transaction(
    Rifath,
    Armeen.identity,
    15.0
)

t1.sign_tran()
transactions.append(t1)

t2 = Transaction(
    Armeen,
    Sara.identity,
    17.0
)

t2.sign_tran()
transactions.append(t2)

t3 = Transaction(
    Sara,
    Armeen.identity,
    10.0
)

#blockchain
TPCoins=[]

block0=Block()
block0.previous_block_hash=None
Nonce=None
block0.verified_transaction.append(t0)
```

```
digest=hash(block0)
last_block_hash = digest
last_block_hash=digest
TPCoins.append(block0)


block1=Block()
block1.previous_block_hash=last_block_hash
block1.verified_transaction.append(t1)
block1.verified_transaction.append(t2)
block1.Nonce=mine(block1,2)
digest=hash(block1)
last_block_hash=digest
TPCoins.append(block1)

block2=Block()
block2.previous_block_hash=last_block_hash
block2.verified_transaction.append(t3)
Nonce=mine(block2,2)
block2.Nonce=mine(block2,2)
digest=hash(block2)
last_block_hash=digest
TPCoins.append(block2)

dump_blockchain(TPCoins)
```

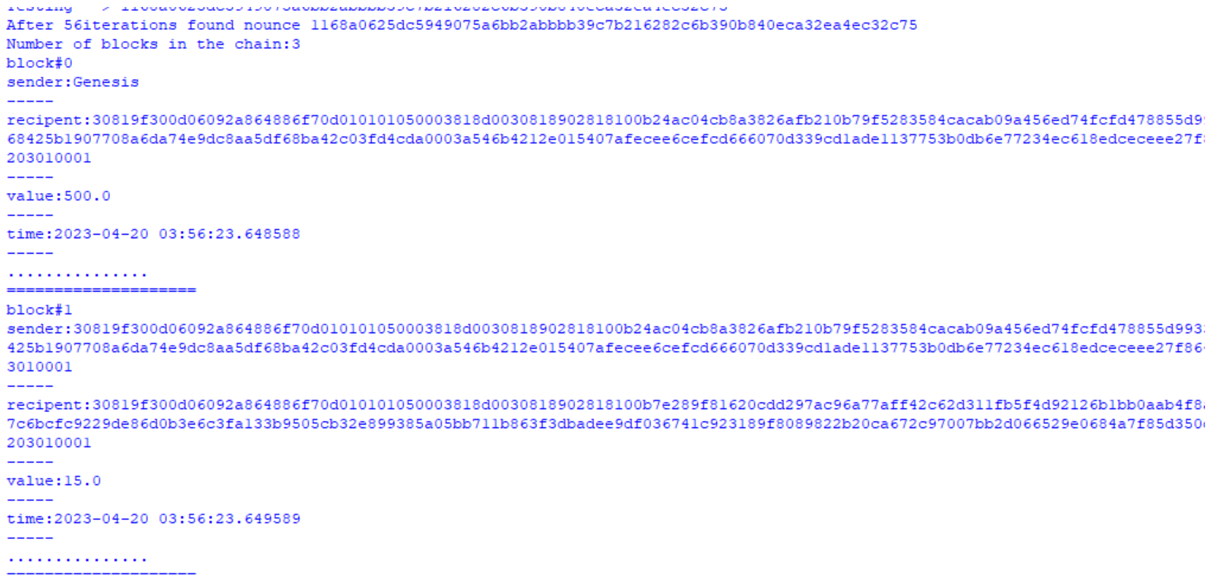
Output:



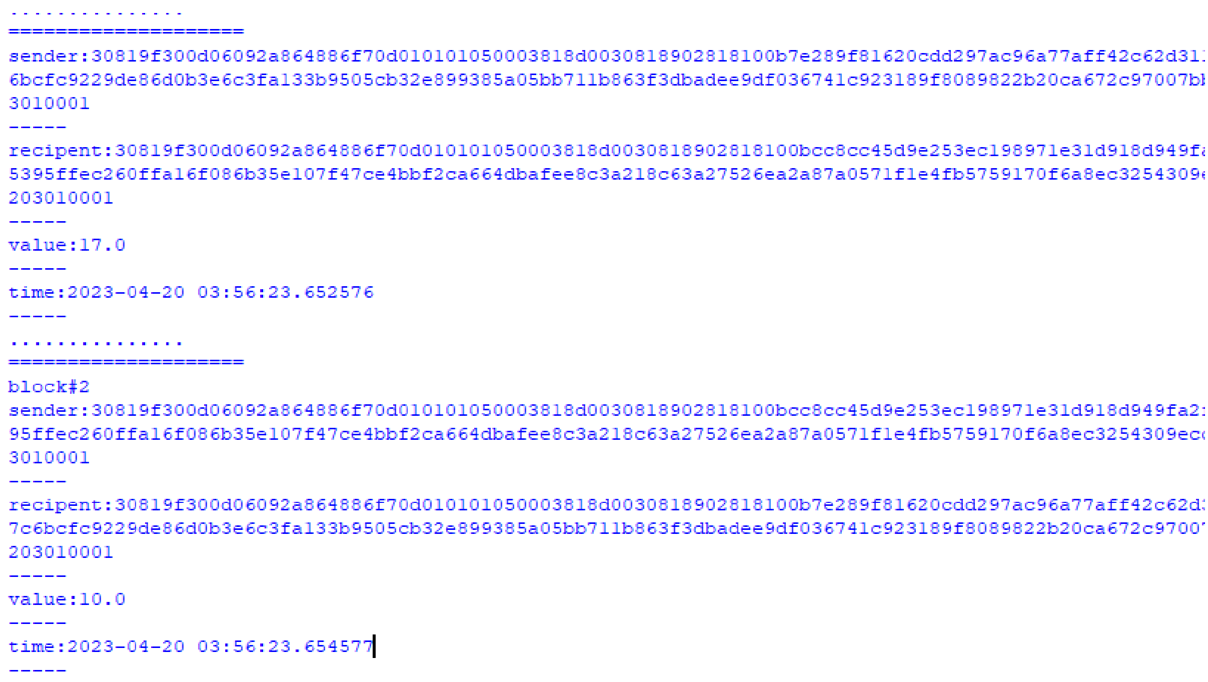
```
============================================================================================ RESTART: C:/Users/arsha/blockchain/prac3a.py ===
======================================================================
Rifath,3
prefix 111
Testing --> 9663f9ef9867d30958fbe492bc67fe33d7f13637628884f35b0b7387326a1f06
Testing --> 2af26dd05395e19b5761463f09f22b1a8063887361d29c406bb1ac1c877d4755
Testing --> 76e35ecb0e034a434f2eef033ba5192f96e1a40fcf97e70560cdfcd4de5d8ab2
Testing --> c9abcb71c9cdac4797d2ba0c710c2598a695d70bcc3cf7309e1d976ac7ac468c
Testing --> 2952367650291a62f86409b92baebb9e598bfa29eaf558716d13259bbc21eddf
Testing --> 59a8f70aee0759c4b752cfbb6f3e12645ef778c71fae802b648a92a2d9238b42
Testing --> d2ded8fbddf21150c31b2cf5f9c2c5144968654801bd53af4c8d50b371f222d0
Testing --> 8de197364275d2701eac232c76a84d4cd1800c7154bda8adcafc594742e03a59
```



```
Testing --> 1168a0625dc5949075a6bb2abbbb39c7b216282c6b390b840eca32ea4ec32c75
After 56iterations found nounce 1168a0625dc5949075a6bb2abbbb39c7b216282c6b390b840eca32ea4ec32c75
prefix 11
Testing --> 9e3361080864588fd58a109d15c8062bf723118f51cb86f88c9ee06a135eea54
Testing --> 9b32d74d4612b3cad4bd2c97ee6f65f5103c632934b7ce24bf6c78f07a227c22
Testing --> ba25be857b1034fe457620845576aa0d16fef898b7a12fb83d28f168bdc54f24
Testing --> 19e95eda815c2001c01c1efb7c639f777245174153a4f13df08168d7aea01c09
```



```
After 56iterations found nounce 1168a0625dc5949075a6bb2abbbb39c7b216282c6b390b840eca32ea4ec32c75
Number of blocks in the chain:3
block#0
sender:Genesis
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100b24ac04cb8a3826afb210b79f5283584cacab09a456ed74fcfd478855d9
68425b1907708a6da74e9dc8aa5df68ba42c03fd4cda0003a546b4212e015407afecee6cefcd666070d339cd1ade1137753b0db6e77234ec618edceceee27f
203010001
-----
value:500.0
-----
time:2023-04-20 03:56:23.648588
-----
...............
====================
block#1
sender:30819f300d06092a864886f70d010101050003818d0030818902818100b24ac04cb8a3826afb210b79f5283584cacab09a456ed74fcfd478855d993
425b1907708a6da74e9dc8aa5df68ba42c03fd4cda0003a546b4212e015407afecee6cefcd666070d339cd1ade1137753b0db6e77234ec618edceceee27f86
3010001
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100b7e289f81620cdd297ac96a77aff42c62d311fb5f4d92126b1bb0aab4f8
7c6bcfc9229de86d0b3e6c3fa133b9505cb32e899385a05bb711b863f3dbadee9df036741c923189f8089822b20ca672c97007bb2d066529e0684a7f85d350
203010001
-----
value:15.0
-----
time:2023-04-20 03:56:23.649589
-----
...............
--------------------
```



```
................
====================
sender:30819f300d06092a864886f70d010101050003818d0030818902818100b7e289f81620cdd297ac96a77aff42c62d31
6bcfc9229de86d0b3e6c3fa133b9505cb32e899385a05bb711b863f3dbadee9df036741c923189f8089822b20ca672c97007b
3010001
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100bcc8cc45d9e253ec198971e31d918d949f
5395ffec260ffa16f086b35e107f47ce4bbf2ca664dbafee8c3a218c63a27526ea2a87a0571f1e4fb5759170f6a8ec3254309
203010001
-----
value:17.0
-----
time:2023-04-20 03:56:23.652576
-----
................
====================
block#2
sender:30819f300d06092a864886f70d010101050003818d0030818902818100bcc8cc45d9e253ec198971e31d918d949fa2
95ffec260ffa16f086b35e107f47ce4bbf2ca664dbafee8c3a218c63a27526ea2a87a0571f1e4fb5759170f6a8ec3254309ec
3010001
-----
recipent:30819f300d06092a864886f70d010101050003818d0030818902818100b7e289f81620cdd297ac96a77aff42c62d
7c6bcfc9229de86d0b3e6c3fa133b9505cb32e899385a05bb711b863f3dbadee9df036741c923189f8089822b20ca672c9700
203010001
-----
value:10.0
-----
time:2023-04-20 03:56:23.654577
-----
```

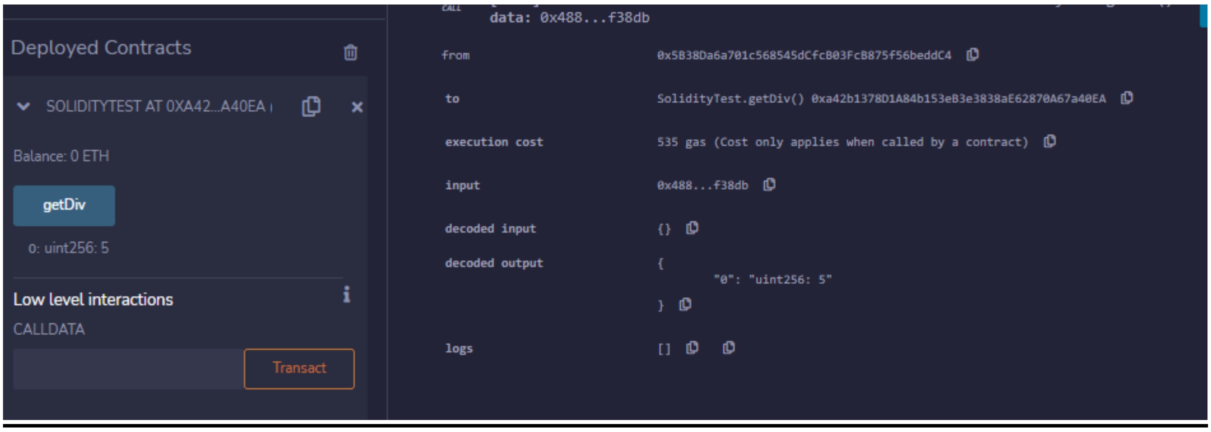
PRACTICAL 4

1. Variable

Code:

```
pragma solidity ^0.8.0;
//RIFATH 3
contract SolidityTest {
   uint storedData; // State variable
   constructor() public{
      storedData=10;
   }
   function getDiv() public view returns(uint){
      uint a=10; // local variable
      uint b=2;
      uint result = a / b;
   return result; // accesss the state variable
}}
```

Output:



```
0: uint256: 5
```

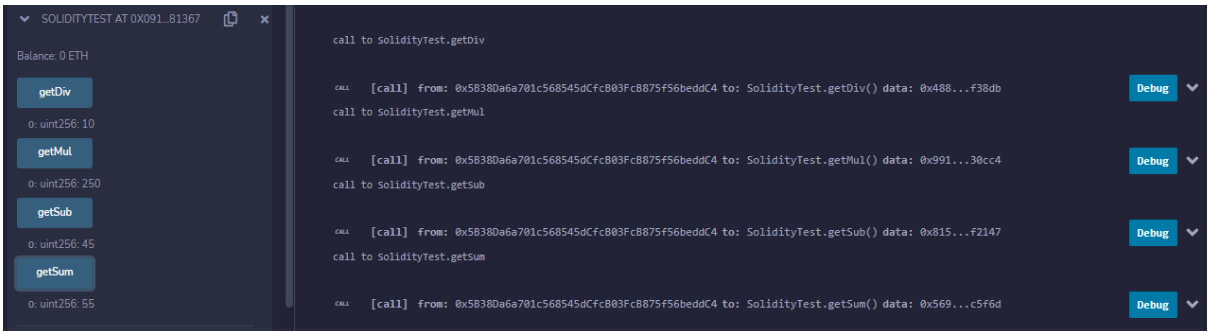
```
     data: 0x488...f38db

from               0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                 SolidityTest.getDiv() 0xa42b1378D1A84b153eB3e3838aE62870A67a40EA
execution cost     535 gas (Cost only applies when called by a contract)
input              0x488...f38db
decoded input      {}
decoded output     {
                        "0": "uint256: 5"
                   }
logs               []
```

2. Operations

Code:

```
pragma solidity ^0.8.0;
//RIFATH 3
contract SolidityTest {
   uint storedData; // State variable
   constructor() public{
      storedData=10;
   }
   function getDiv() public view returns(uint){
      uint a=50; // local variable
      uint b=5;
      uint result = a / b;
   return result; // accesss the state variable
   }
   function getMul() public view returns(uint){
      uint a=50; // local variable
      uint b=5;
      uint result = a * b;
   return result; // accesss the state variable
   }
   function getSum() public view returns(uint){
      uint a=50; // local variable
      uint b=5;
      uint result = a + b;
   return result; // accesss the state variable
   }
   function getSub() public view returns(uint){
      uint a=50; // local variable
      uint b=5;
      uint result = a - b;
   return result; // accesss the state variable
   }}
```

Output:



```
Balance: 0 ETH
getDiv
0: uint256: 10
getMul
0: uint256: 250
getSub
0: uint256: 45
getSum
0: uint256: 55
```

```
call to SolidityTest.getDiv

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SolidityTest.getDiv() data: 0x488...f38db
call to SolidityTest.getMul

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SolidityTest.getMul() data: 0x991...30cc4
call to SolidityTest.getSub

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SolidityTest.getSub() data: 0x815...f2147
call to SolidityTest.getSum

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SolidityTest.getSum() data: 0x569...c5f6d
```

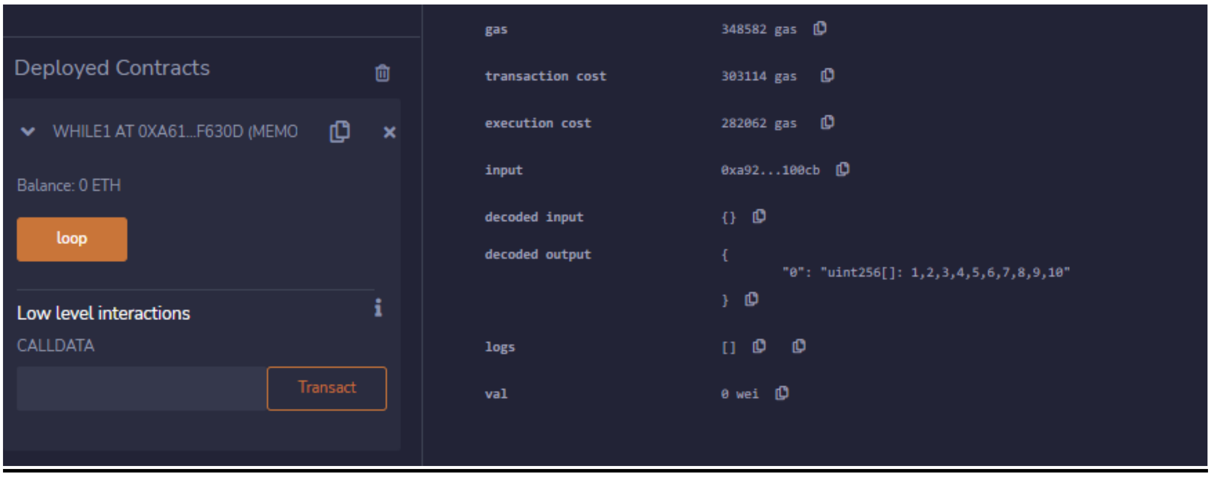
3. Loops

a. While

Code:

```
pragma solidity ^0.8.0;
//rifath 3
contract while1{
uint[] data;
uint8 j=0;
function loop() public returns(uint[] memory)
{
   while (j<10)
   {
      j++;
      data.push(j);
   }
   return data;
}
}
}
```

Output:



Balance: 0 ETH

```
gas                 348582 gas
transaction cost    303114 gas
execution cost      282062 gas
input               0xa92...100cb
decoded input       {}
decoded output      {
                        "0": "uint256[]: 1,2,3,4,5,6,7,8,9,10"
                    }
logs                []
val                 0 wei
```

Do While

Code:

```
pragma solidity ^0.8.0;
//rifath 3
contract doWhile1{
uint[] data;
uint8 j=0;
function loop() public returns(uint[] memory)
{
    do
    {
        j++;
        data.push(j);
    }
    while (j<10);
    return data;
}
}
```

b. For

Code:

```
pragma solidity ^0.8.0;
contract ForLoop{
    function count() public pure returns(uint256){
        uint256 sum=0;
        for(uint256 i=0;i<=25;i++){
            sum+=i;
        }
        return sum;
   }}
```

Output:

4. Decision Making

a. If else

Code:

```
pragma solidity ^0.8.0;
contract Check{
   uint i=100;
   uint j=80;
   function ifElse() public returns(string memory)
   {
      if(i<j)
      {
         return "i is smaller than j";
      }
      else
            {
         return " i is greater than j";
      }  }}
```

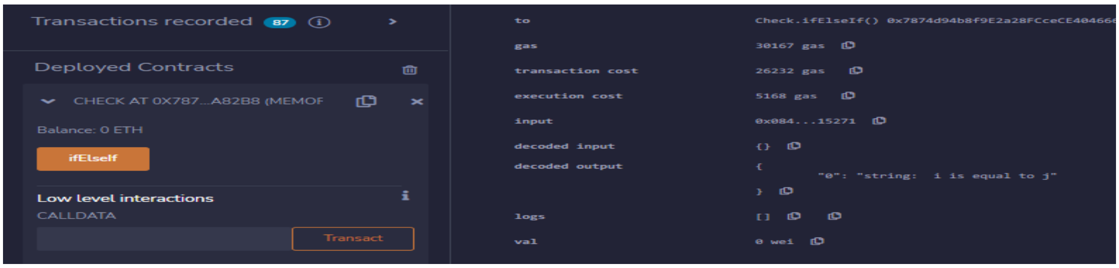
Output:

b. If else-if

Code:

```
pragma solidity ^0.8.0;
contract Check{
   uint i=100;
   uint j=100;
   function ifElseIf() public returns(string memory)
   {
      if(i<j)
      {
         return "i is smaller than j";}
      else if(i>j)
      {
         return " i is greater than j"; }
      else
      {
         return " i is equal to j";
     }}}
```

Output:



```
to                    Check.ifElseIf() 0x7874d94b8f9E2a28FCceCE404666
gas                   30167 gas
transaction cost      26232 gas
execution cost        5168 gas
input                 0x084...15271
decoded input         {}
decoded output        {
                              "0": "string:  i is equal to j"
                      }
logs                  []
val                   0 wei
```

5. Strings

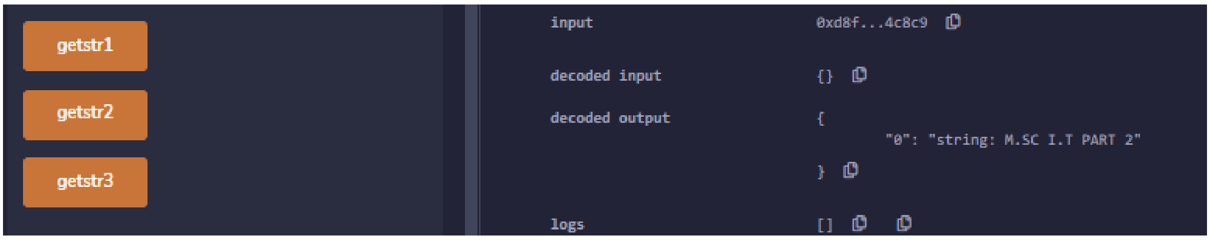
a. Regular string

Code:

```
pragma solidity ^0.8.0;
contract SS{
   string str1="M.SC I.T PART 2";
   string str2='K.C COLLEGE, COLABA';
   string str3=new string(20);
   function getstr1() public returns(string memory)
   {
      return str1;   }
   function getstr2() public returns(string memory)
   {
      return str2;  }
   function getstr3() public returns(string memory)
   {
      return str3;
   }}
```

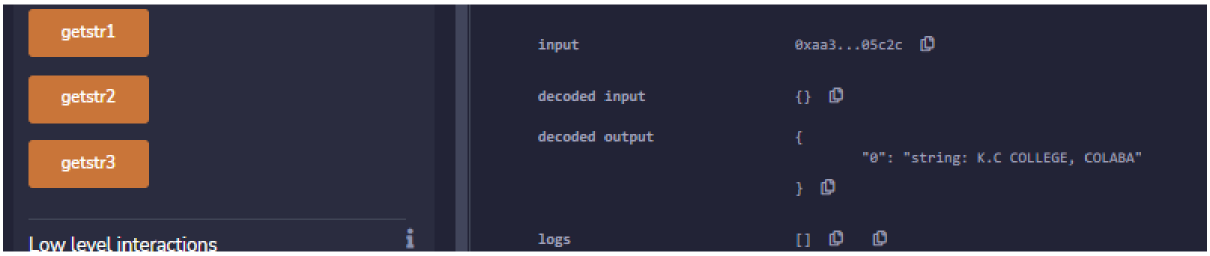
Output:

String1:



```
input             0xd8f...4c8c9
decoded input     {}
decoded output    {
                          "0": "string: M.SC I.T PART 2"
                  }
logs              []
```

String2:



```
input             0xaa3...05c2c
decoded input     {}
decoded output    {
                          "0": "string: K.C COLLEGE, COLABA"
                  }
logs              []
```

String3:

b. Concatenate

Code:

```
pragma solidity >=0.5.0 <0.9.0;
//rifath 3
  contract Demo{
    string public s1 = "RIFATH ";
    string public s2 = "ZAHRAA";
    string public new_str;

    function concatenate() public {
      new_str = string(abi.encodePacked(s1, s2));
     }
}
```

Output:

c. Compare

Code:

```
pragma solidity ^0.8.0;
contract Demo{
   string str1="rifath";
   string str2='rifath”';
   bool public isEqual;
   function cmp() public
   {
      isEqual=keccak256(abi.encodePacked(str1))==keccak256(abi.encodePacked(str2));
   }
}
```

Output:

PRACTICAL 5

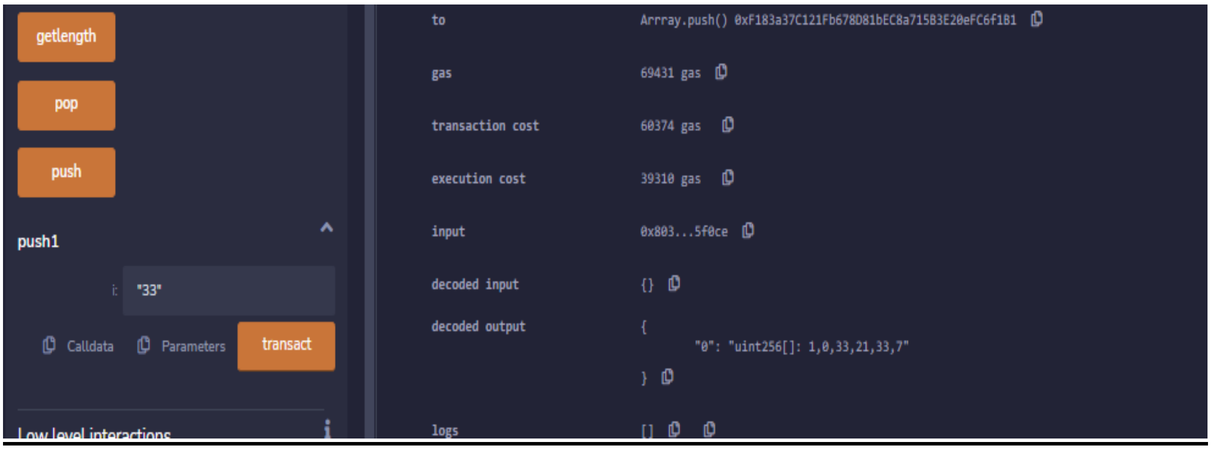
1. Arrays

Code:

```
pragma solidity ^0.5.0;
contract Arrray{
   uint[] nums=[1,2,33,21];

   function getlength() public returns(uint){
      return nums.length;
   }

   function pop() public{
      delete nums[1];
   }
   function push() public returns (uint[] memory){
      nums.push(7);
      return nums;
   }
function push1(uint i) public{
         nums.push(i);
      }
}
```

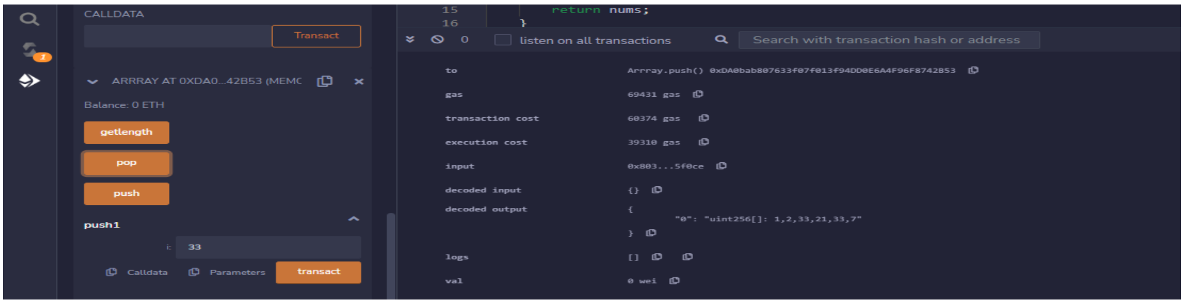
Output:

Push:



```
to                  Arrray.push() 0xF183a37C121Fb678D81bEC8a715B3E20eFC6f1B1
gas                 69431 gas
transaction cost    60374 gas
execution cost      39310 gas
input               0x803...5f0ce
decoded input       {}
decoded output      {
                          "0": "uint256[]: 1,0,33,21,33,7"
                    }
logs                []
```

Pop:

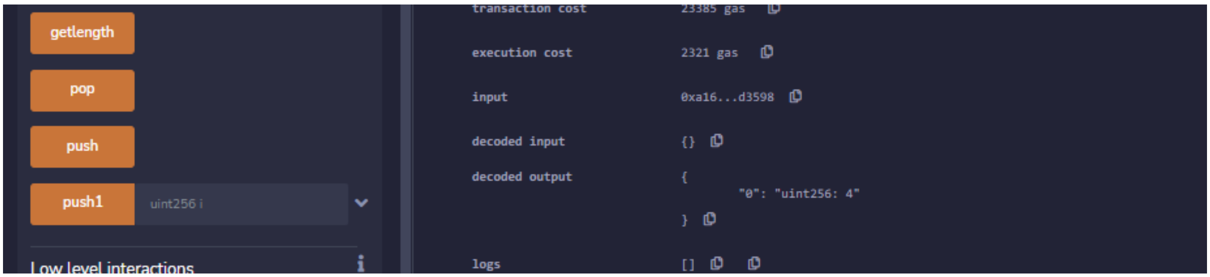


CALLDATA
Transact
ARRRAY AT 0XDA0...42B53 (MEMC
Balance: 0 ETH
getlength
pop
push
push1
i: 33
Calldata
Parameters
transact
listen on all transactions
Search with transaction hash or address

```
        return nums;
    }
```

```
to                  Arrray.push() 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
gas                 69431 gas
transaction cost    60374 gas
execution cost      39310 gas
input               0x803...5f0ce
decoded input       {}
decoded output      {
                          "0": "uint256[]: 1,2,33,21,33,7"
                    }
logs                []
val                 0 wei
```

Dynamic:



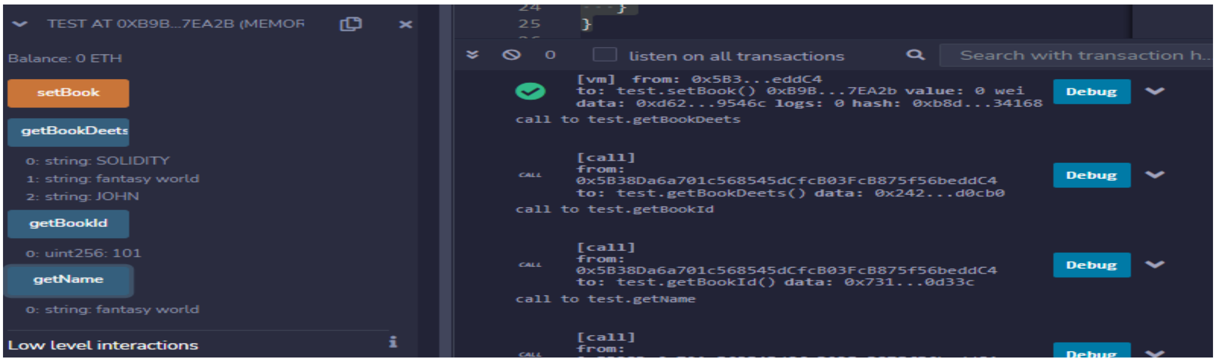
```
transaction cost        23385 gas
execution cost          2321 gas
input                   0xa16...d3598
decoded input           {}
decoded output          {
                            "0": "uint256: 4"
                        }
logs                    []
```

2. Struct

Code:

```
pragma solidity ^0.5.0;
contract test{
   struct Book{
      string title;
      string author;
      string name;
      uint book_id;
   }
  Book;
  function setBook() public{
     book = Book('SOLIDITY','JOHN','fantasy world',101);
  }

  function getBookId() public view returns(uint){
      return book.book_id;
  }

   function getName() public view returns(string memory){
      return book.name;
  }

  function getBookDeets() public view returns(string memory,string memory){
     return(book.title,book.name,book.author);
     }
}
```

Output:


TEST AT 0XB9B...7EA2B (MEMOR
setBook
getBookDeets
getBookId
getName
Low level interactions
listen on all transactions
Search with transaction h...
Debug

```
Balance: 0 ETH
0: string: SOLIDITY
1: string: fantasy world
2: string: JOHN
0: uint256: 101
0: string: fantasy world
```

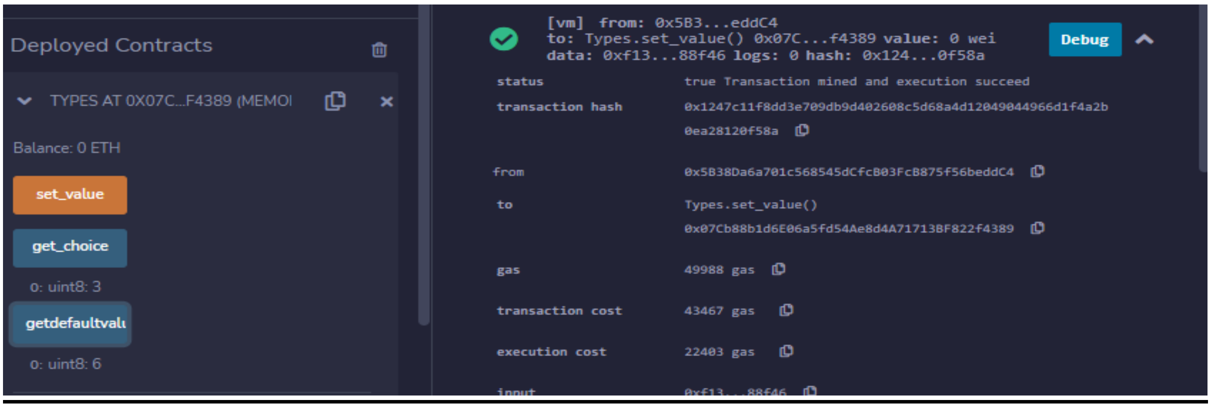
```
25    }

[vm]  from: 0x5B3...eddC4
to: test.setBook() 0xB9B...7EA2b value: 0 wei
data: 0xd62...9546c logs: 0 hash: 0xb8d...34168
call to test.getBookDeets

[call]
from:
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to: test.getBookDeets() data: 0x242...d0cb0
call to test.getBookId

[call]
from:
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to: test.getBookId() data: 0x731...0d33c
call to test.getName

[call]
from:
```

3. Enum

Code:

```
pragma solidity ^0.5.0;

contract Types{

      enum week_days
      {
         Monday,
         Tuesday,
         Wednesday,
         Thursday,
         Friday,
         Saturday,
         Sunday
      }
      week_days week;

      week_days choice;

      week_days constant default_value = week_days.Sunday;

      function set_value() public{
         choice = week_days.Thursday;
      }

      function get_choice() public view returns(week_days){
         return choice;
      }
      function getdefaultvalue() public pure returns(week_days){
         return default_value;
      }
}
```

Output:



```
Balance: 0 ETH
0: uint8: 3
0: uint8: 6
```

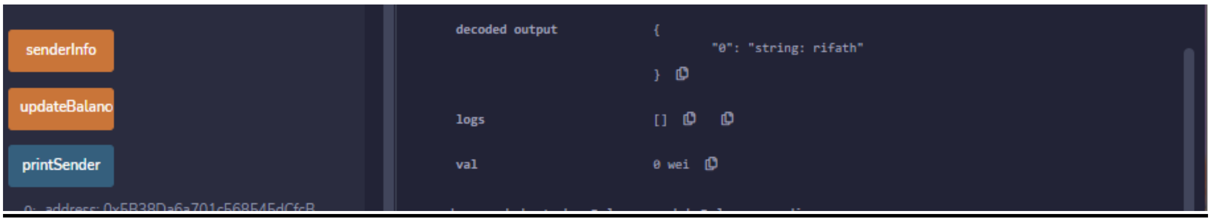
```
[vm]  from: 0x5B3...eddC4
to: Types.set_value() 0x07C...f4389 value: 0 wei
data: 0xf13...88f46 logs: 0 hash: 0x124...0f58a
status              true Transaction mined and execution succeed
transaction hash    0x1247c11f8dd3e709db9d402608c5d68a4d12049044966d1f4a2b
                    0ea28120f58a
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  Types.set_value()
                    0x07Cb88b1d6E06a5fd54Ae8d4A71713BF822f4389
gas                 49988 gas
transaction cost    43467 gas
execution cost      22403 gas
input               0xf13...88f46
```

4. Mapping

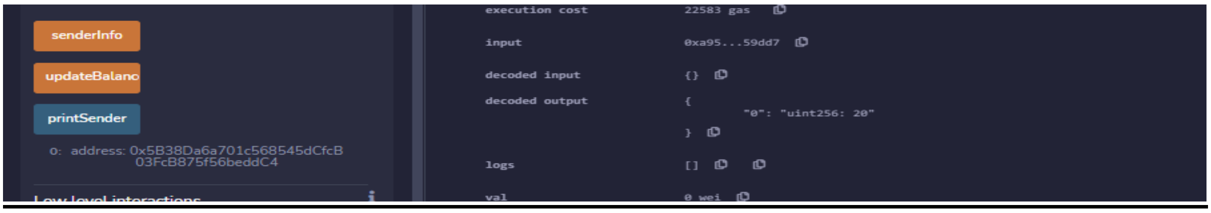
Code:

```
pragma solidity ^0.5.0;
contract LedgerBalance{
   mapping(address => uint) balance;
   mapping(address => string) name;
   function updateBalance() public returns(uint){
         balance[msg.sender]=20;
         return balance[msg.sender];
   }

   function senderInfo() public returns(string memory){
      name[msg.sender] = "rifath";
      return name[msg.sender];
   }
   function printSender() public view returns(address){
      return msg.sender;
   }
}
```

Output:

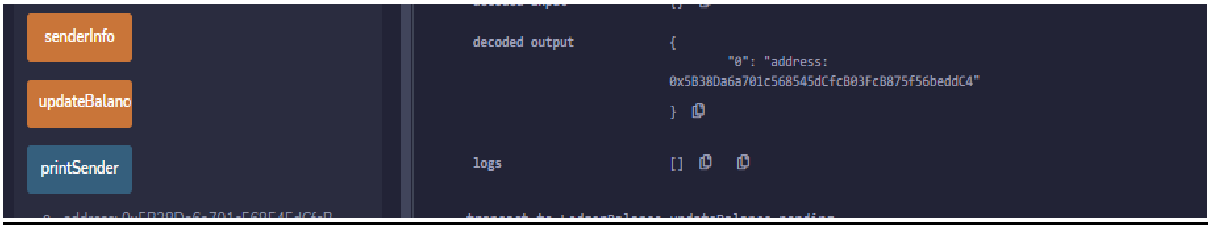


```
decoded output        {
                            "0": "string: rifath"
                      }
logs                  []
val                   0 wei
```



0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

```
execution cost        22583 gas
input                 0xa95...59dd7
decoded input         {}
decoded output        {
                            "0": "uint256: 20"
                      }
logs                  []
val                   0 wei
```



```
decoded output        {
                            "0": "address:
                      0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
                      }
logs                  []
```

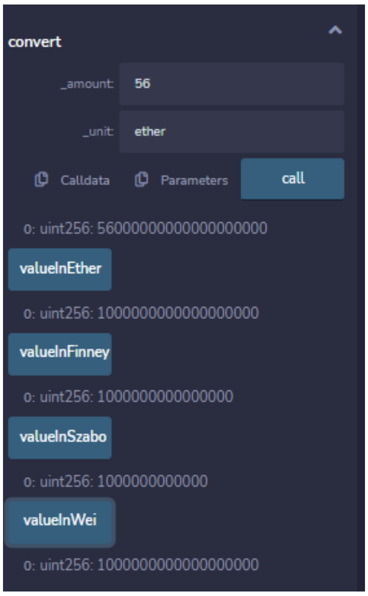
6. Conversions and Ether units

keccak256

Code:

```
pragma solidity >=0.4.0 <0.7.0;
contract EtherUnitsExample {
   uint256 public valueInWei = 1 ether; // 1 ether in Wei
   uint256 public valueInFinney = 1 finney; // 1 finney in Wei
   uint256 public valueInSzabo = 1 szabo; // 1 szabo in Wei
   uint256 public valueInEther = 1 ether; // 1 ether in Wei

   function convert(uint256 _amount, string memory _unit) public pure returns (uint256) {
      if (keccak256(abi.encodePacked(_unit)) == keccak256(abi.encodePacked("wei"))) {
         return _amount;
      } else if (keccak256(abi.encodePacked(_unit)) == keccak256(abi.encodePacked("finney")))
{
         return _amount * 1 finney;
      } else if (keccak256(abi.encodePacked(_unit)) == keccak256(abi.encodePacked("szabo")))
{
         return _amount * 1 szabo;
      } else if (keccak256(abi.encodePacked(_unit)) == keccak256(abi.encodePacked("ether"))
|| keccak256(abi.encodePacked(_unit)) == keccak256(abi.encodePacked("eth"))) {
         return _amount * 1 ether;
      } else {
         revert("Invalid unit");
      }
   }}
```

Output:



```
_amount: 56
_unit: ether
0: uint256: 56000000000000000000
valueInEther
0: uint256: 1000000000000000000
valueInFinney
0: uint256: 1000000000000000
valueInSzabo
0: uint256: 1000000000000
valueInWei
0: uint256: 1000000000000000000
```

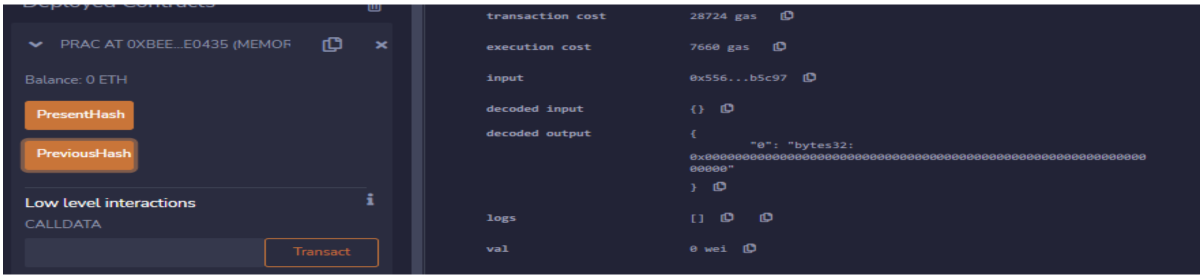
7. Special variables:

a. Solidity contract to demonstrate the special variables block.number and blockhash.

Code:

```
pragma solidity ^0.5.0;
contract prac
{
   uint BNumber;
   bytes32 BHashPresent;
   bytes32 BHashPrevious;
   function PresentHash()
         public returns(bytes32)
   {
      BNumber = block.number;
      BHashPresent =blockhash(BNumber);
      return BHashPresent;
   }
   function PreviousHash()
         public returns(bytes32)
   {
      BNumber = block.number;
      BHashPrevious = blockhash(BNumber - 1);
      return BHashPrevious;
   }
}
```

Output:



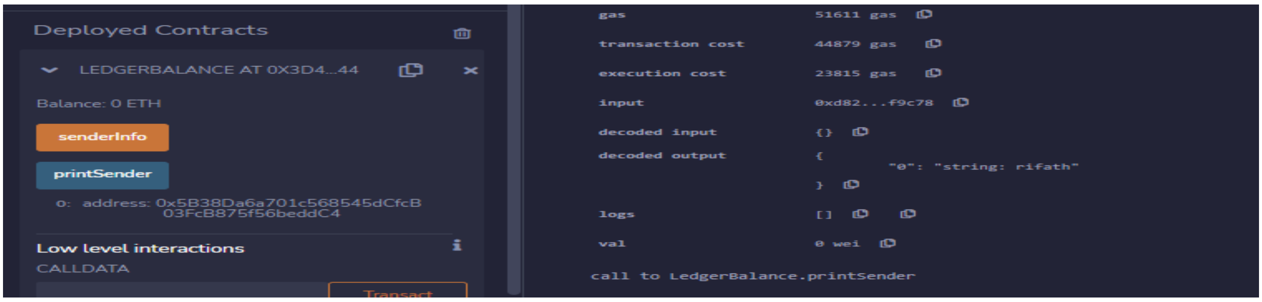
```
transaction cost      28724 gas
execution cost        7660 gas
input                 0x556...b5c97
decoded input         {}
decoded output        {
                              "0": "bytes32:
                      0x00000000000000000000000000000000000000000000000000000000000
                      00000"
                      }
logs                  []
val                   0 wei
```

b. Solidity contract to demonstrate msg.sender

Code:

```
pragma solidity ^0.5.0;
contract LedgerBalance{
   mapping(address => string) name;
   function senderInfo() public returns(string memory){
      name[msg.sender] = "rifath";
      return name[msg.sender];
   }

   function printSender() public view returns(address){
      return msg.sender;
   }
}
```

Output:



```
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
```

```
gas                     51611 gas
transaction cost        44879 gas
execution cost          23815 gas
input                   0xd82...f9c78
decoded input           {}
decoded output          {
                              "0": "string: rifath"
                        }
logs                    []
val                     0 wei

call to LedgerBalance.printSender
```

PRACTICAL 6

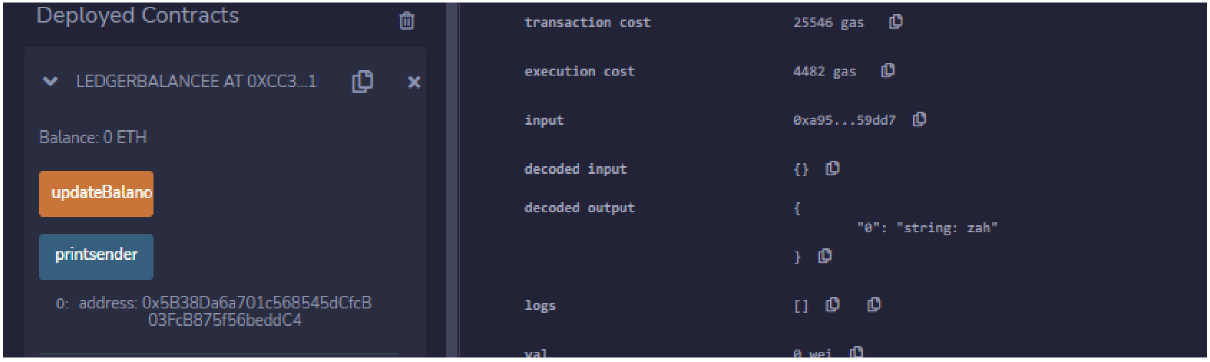
Implement and demonstrate the use of the following in Solidity

1. **Functions**

Code:

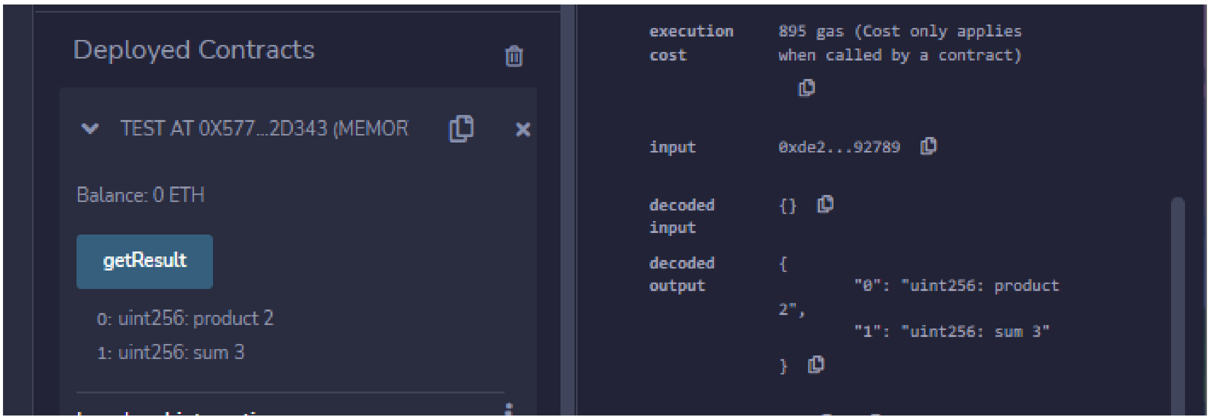
```
pragma solidity ^0.8.0;
contract LedgerBalancee {
   mapping(address => string) name;
   function updateBalance() public returns(string memory) {
      name[msg.sender]="zah";
      return name [msg.sender];
   }
   function printsender() public view returns(address){
      return msg.sender;
   }
}
contract Test {
  function getResult() public view returns(uint product, uint sum){
     uint a = 1; // local variable
     uint b = 2;
     product = a * b;
     sum = a + b;
     //return(a*b, a+b);
  }}
```

Output:



```
0: address: 0x5B38Da6a701c568545dCfcB
            03FcB875f56beddC4
```

```
transaction cost        25546 gas

execution cost          4482 gas

input                   0xa95...59dd7

decoded input           {}

decoded output          {
                                "0": "string: zah"
                        }

logs                    []

val                     0 wei
```



```
0: uint256: product 2
1: uint256: sum 3
```

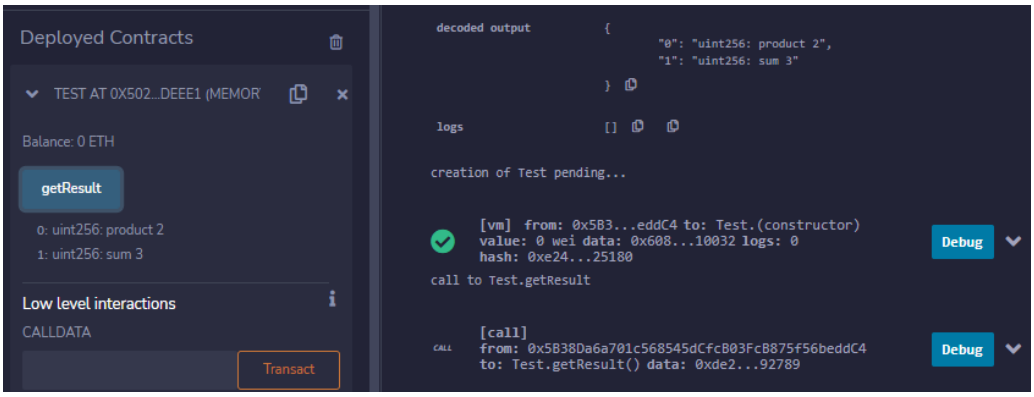
```
execution     895 gas (Cost only applies
cost          when called by a contract)

input         0xde2...92789

decoded       {}
input

decoded       {
output                "0": "uint256: product
              2",
                      "1": "uint256: sum 3"
              }
```

2. .View Functions

Code:

```
pragma solidity ^0.5.0;
contract Test{
   function getResult() public view  returns(uint product, uint sum){
      uint a=1;// local variable
      uint b=2;
      product=a*b;
      sum = a+b;
   }
}
```

Output:



```
0: uint256: product 2
1: uint256: sum 3
```

```
decoded output          {
                                "0": "uint256: product 2",
                                "1": "uint256: sum 3"
                        }

logs                    []

creation of Test pending...

       [vm]  from: 0x5B3...eddC4 to: Test.(constructor)
       value: 0 wei data: 0x608...10032 logs: 0
       hash: 0xe24...25180
call to Test.getResult

       [call]
CALL   from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
       to: Test.getResult() data: 0xde2...92789
```

3. .Pure Functions

Code:

```
pragma solidity ^0.5.0;
contract Test{
   function getResult() public pure returns(uint product,uint sum){
      uint a = 1; //local variable
      uint b = 2;
      product = a*b;
      sum = a+b;
   }
}
```

Output:

4. .Fallback Functions

Code:

```
pragma solidity ^0.5.12;

contract A {
   uint n;
   function set(uint value) external {
      n=value;
   }

   //fallback function
   function() external payable{
      n=0;
   }
}
contract example{
   function callA(A a) public returns (bool){

      (bool success,) = address(a).call(abi.encodeWithSignature("setter()"));
      require(success);

      address payable payableA=address(uint160(address(a)));
      return(payableA.send(2 ether));
   }
}
```

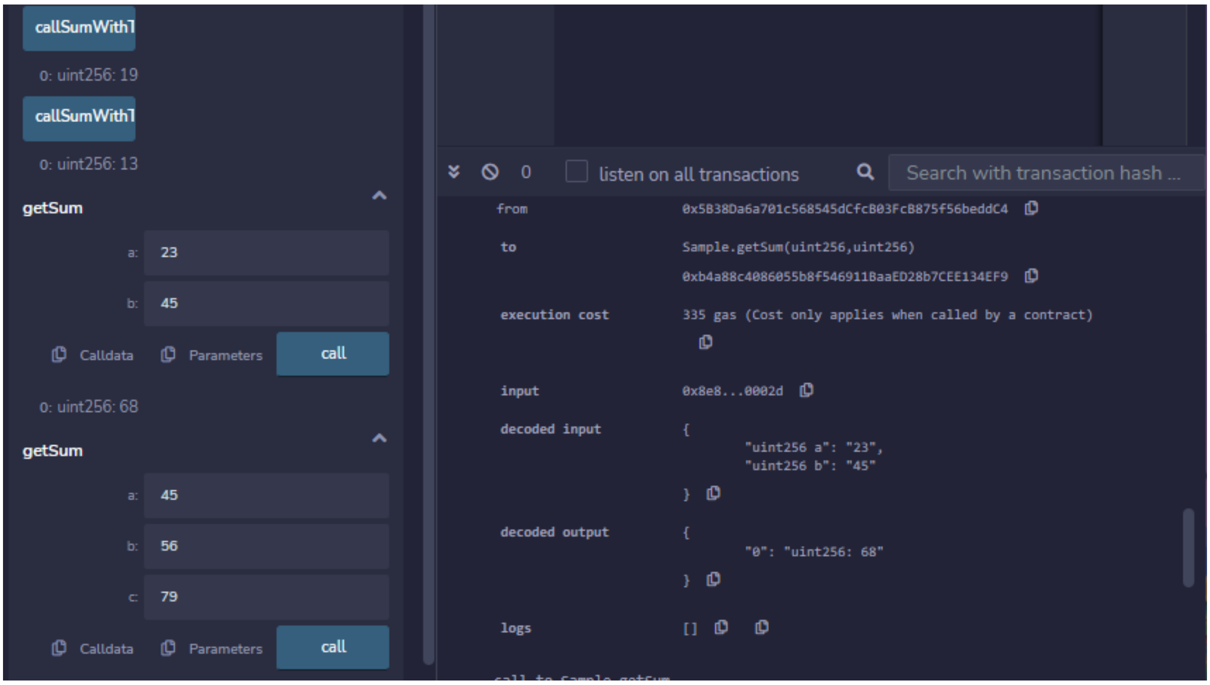
Output:

5. .Function Overloading

Code:

```
pragma solidity ^0.5.12;

contract Sample{
   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(4,9);
   }

    function callSumWithThreeArguments() public pure returns (uint){
      return getSum(4,9,6);
   }
}
```

Output:



```
0: uint256: 19
0: uint256: 13
a: 23
b: 45
0: uint256: 68
a: 45
b: 56
c: 79
```

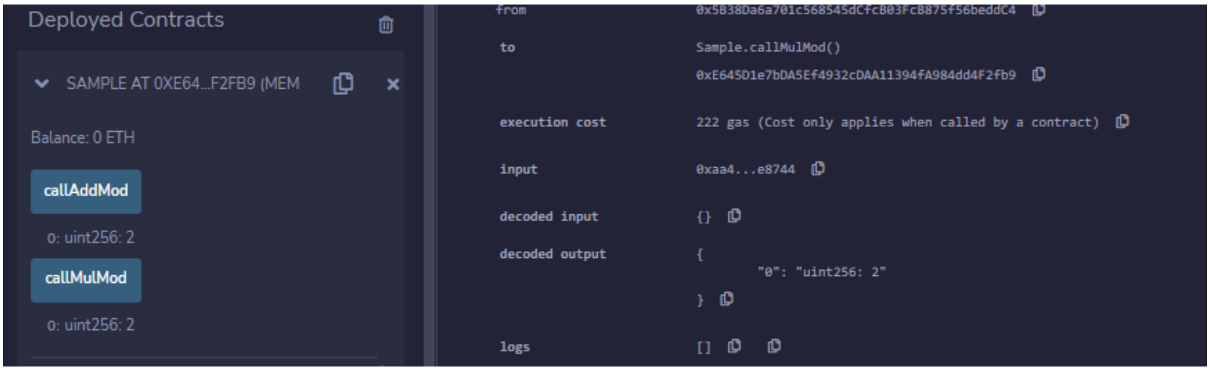
```
from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                Sample.getSum(uint256,uint256)
                  0xb4a88c4086055b8f546911BaaED28b7CEE134EF9
execution cost    335 gas (Cost only applies when called by a contract)

input             0x8e8...0002d
decoded input     {
                        "uint256 a": "23",
                        "uint256 b": "45"
                  }
decoded output    {
                        "0": "uint256: 68"
                  }
logs              []
```

6. .Mathematical Functions

Code:

```
pragma solidity ^0.5.0;

contract Sample{
    function callAddMod() public pure returns (uint){
        return addmod(3,4,5);
//3+4 % 5
    }

   function callMulMod() public pure returns (uint){
        return mulmod(3,4,5);
    }
//3*4 % 5
}
```

Output:



```
0: uint256: 2
0: uint256: 2
```

```
from                  0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                    Sample.callMulMod()
                      0xE645D1e7bDA5Ef4932cDAA11394fA984dd4F2fb9
execution cost        222 gas (Cost only applies when called by a contract)
input                 0xaa4...e8744
decoded input         {}
decoded output        {
                              "0": "uint256: 2"
                      }
logs                  []
```

7. .Cryptographic Functions

Code:

```
pragma solidity ^0.5.12;
contract Test{
   function callsha256() public pure returns(bytes32 result){
      return sha256("rifath");
   }
   function callkeccak256() public pure returns(bytes32 result){
      return keccak256("rifath");
   }
}
```

Output:

PRACTICAL 7

Implement and demonstrate the use of the following in Solidity

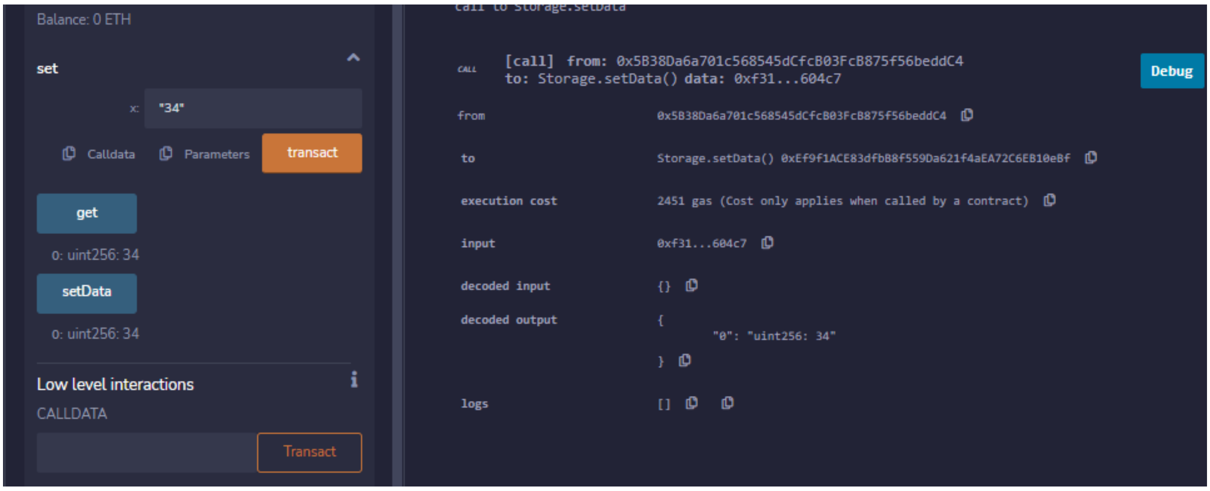
1. Contracts
2. Inheritance
3. Constructors
4. Abstract class
5. Interfaces

1. Contracts

Code:

```
pragma solidity ^0.8.0;
contract Storage
{
uint public setData;
function set(uint x) public{
setData = x;
}
function get() public view returns (uint) {
   return setData;
}}
```

Output:



```
x: "34"
0: uint256: 34
0: uint256: 34
```

```
call to Storage.setData

CALL    [call]  from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
        to: Storage.setData() data: 0xf31...604c7

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                Storage.setData() 0xEf9f1ACE83dfbB8f559Da621f4aEA72C6EB10eBf
execution cost    2451 gas (Cost only applies when called by a contract)
input             0xf31...604c7
decoded input     {}
decoded output    {
                        "0": "uint256: 34"
                  }
logs              []
```

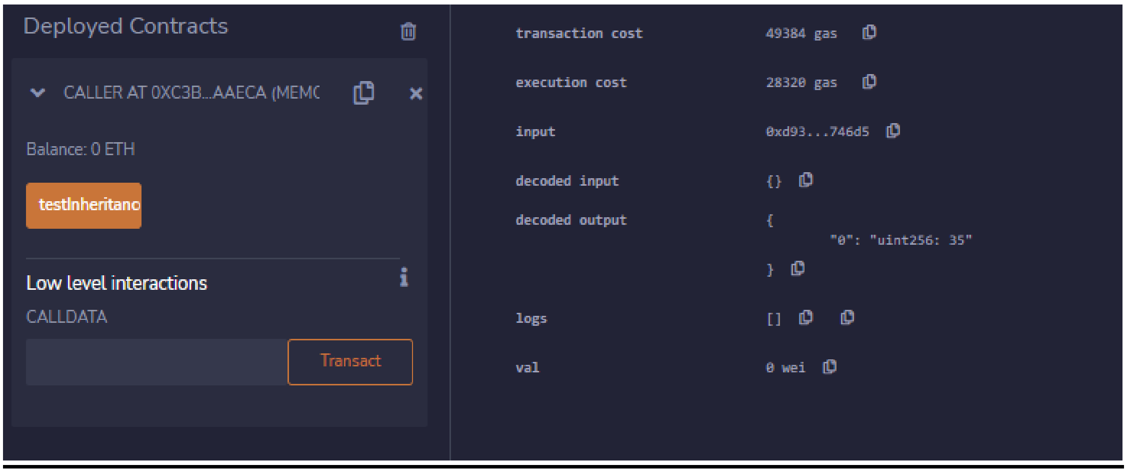
2. Inheritance

a. Single Inheritance:

Code:

```
pragma solidity 0.5.0;
contract parent{

   uint internal sum;

   function setValue() external {
      uint a = 10;
      uint b = 25;
      sum = a + b;
   }
}
contract child is parent{ //defining the child contract

   function getValue(
   ) external view returns(uint) {
      return sum;
   }
}
contract caller {
   child cc = new child();

   function testInheritance(
   ) public returns (uint) {
      cc.setValue();
      return cc.getValue();
   }
}
```

Output:



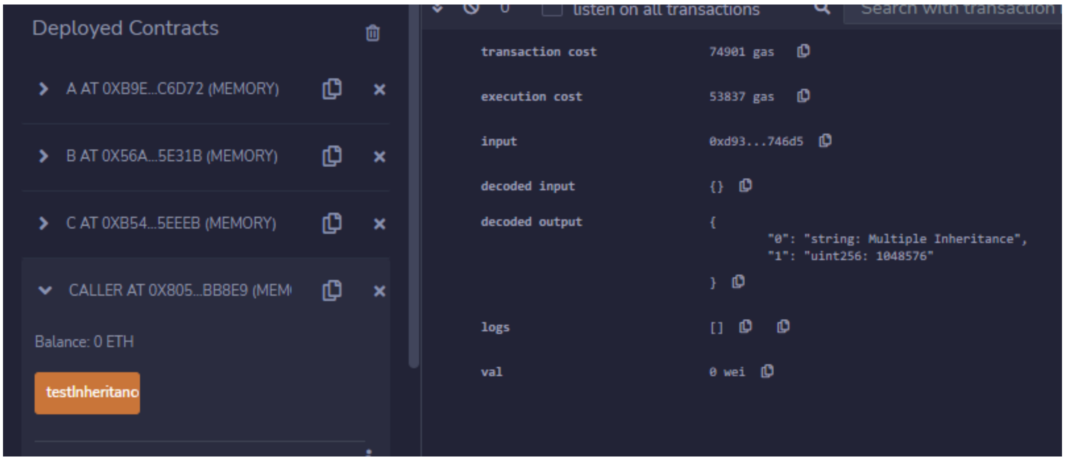
```
transaction cost        49384 gas
execution cost          28320 gas
input                   0xd93...746d5
decoded input           {}
decoded output          {
                                "0": "uint256: 35"
                        }
logs                    []
val                     0 wei
```

b. Multiple Inheritance:

Code:

```
pragma solidity ^0.5.0;

contract A {
    string internal x;

    function setA() external {
        x = "Multiple Inheritance";
    }
}
contract B {
    uint256 internal pow;

    function setB() external {
        uint256 a = 2;
        uint256 b = 20;
        pow = a**b;
    }
}
contract C is A, B {
    function getStr() external view returns (string memory)
    {
        return x;
    }
    function getPow() external view returns (uint256)
    {
        return pow;
    }
}
contract caller {
    C contractC = new C();
    function testInheritance() public returns (string memory, uint256) {
        contractC.setA();
        contractC.setB();
        return (contractC.getStr(), contractC.getPow());
    }
}
```

Output:



Deployed Contracts
A AT 0XB9E...C6D72 (MEMORY)
B AT 0X56A...5E31B (MEMORY)
C AT 0XB54...5EEEB (MEMORY)
CALLER AT 0X805...BB8E9 (MEM
Balance: 0 ETH
testInheritanc
listen on all transactions
Search with transaction

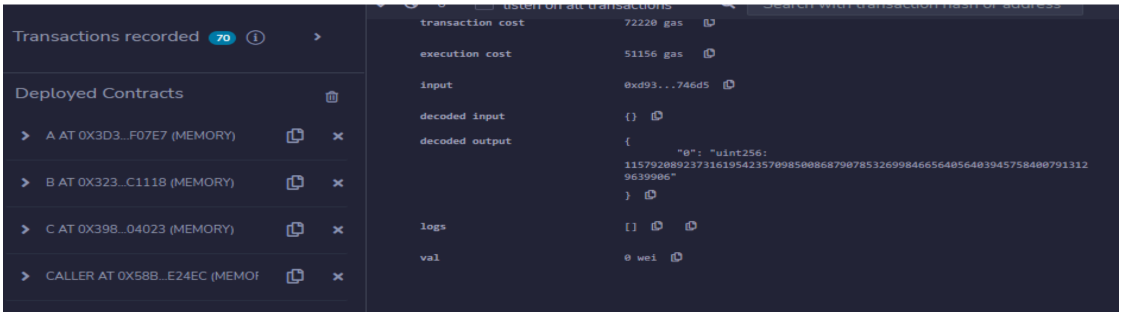
```
transaction cost        74901 gas
execution cost          53837 gas
input                   0xd93...746d5
decoded input           {}
decoded output          {
                                "0": "string: Multiple Inheritance",
                                "1": "uint256: 1048576"
                        }
logs                    []
val                     0 wei
```

c. Multilevel Inheritance:

Code:

```
pragma solidity ^0.5.0;

contract A {
      uint256 internal x;
    function setX() external {
        x=600;
    }
}
contract B is A {
      uint256 internal y;
    function setY() external {
        y=20-x;
    }
}
contract C is B{
    function getY() external view returns(
      uint){
        return y;
    }
}
contract caller {
    C cc = new C();
    function testInheritance(
    ) public returns (
      uint256) {
        cc.setX();
        cc.setY();
        return cc.getY();
    }
}
```

Output:


Transactions recorded 70
Deployed Contracts
A AT 0X3D3...F07E7 (MEMORY)
B AT 0X323...C1118 (MEMORY)
C AT 0X398...04023 (MEMORY)
CALLER AT 0X58B...E24EC (MEMOI

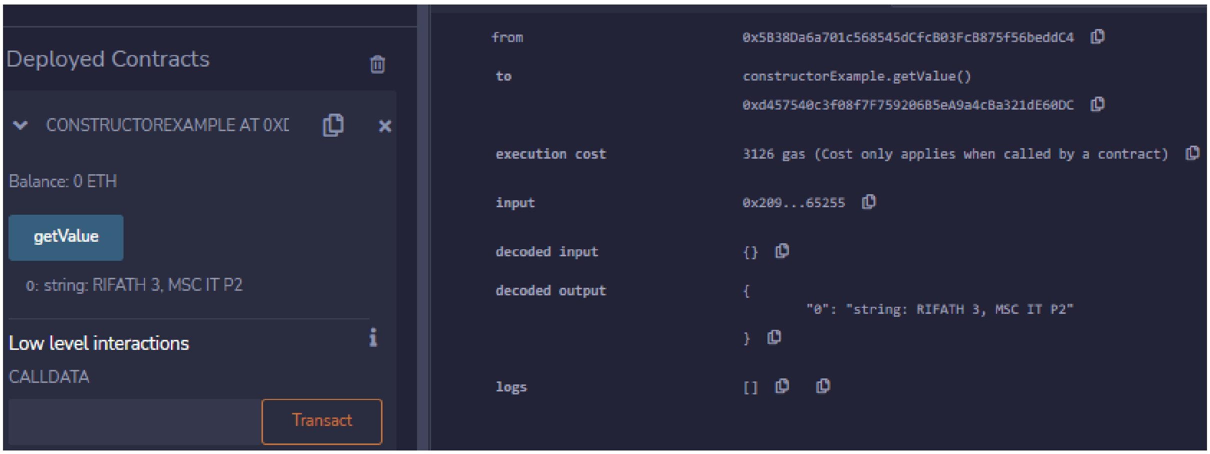
```
transaction cost          72220 gas
execution cost            51156 gas
input                     0xd93...746d5
decoded input             {}
decoded output            {
                                  "0": "uint256:
                          115792089237316195423570985008687907853269984665640564039457584007913129639906"
                          }
logs                      []
val                       0 wei
```

3. Constructors

Code:

```
pragma solidity ^0.5.0;

// Creating a contract
contract constructorExample {

// Declaring state variable
string str;

constructor() public {
str = "RIFATH 3, MSC IT P2";
}
// Defining function to
// return the value of 'str'
function getValue(
) public view returns (
string memory) {
return str;
}
}
```

Output:



```
0: string: RIFATH 3, MSC IT P2
```

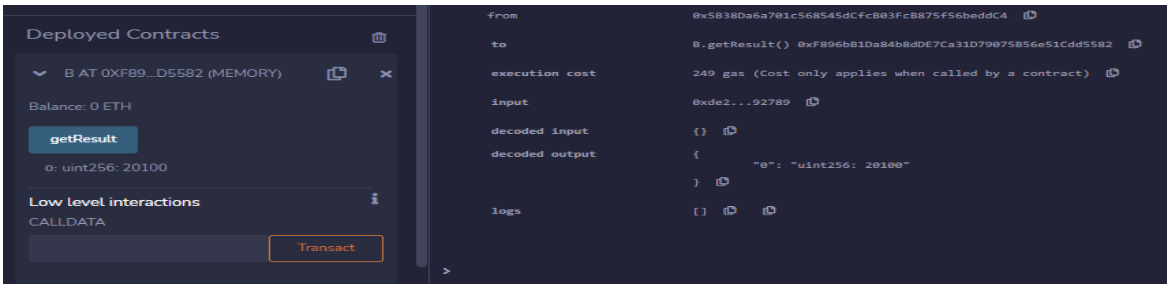
```
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  constructorExample.getValue()
                    0xd457540c3f08f7F759206B5eA9a4cBa321dE60DC
execution cost      3126 gas (Cost only applies when called by a contract)
input               0x209...65255
decoded input       {}
decoded output      {
                            "0": "string: RIFATH 3, MSC IT P2"
                    }
logs                []
```

4. Abstract class

Code:

```
pragma solidity ^0.5.0;

contract A {
   function getResult() public view returns(uint);
}
contract B is A {
   function getResult() public view returns(uint) {
      uint a = 100;
      uint b = 201;
      uint result = a * b;
      return result;
   }
}
```

Output:



```
Balance: 0 ETH
0: uint256: 20100
```

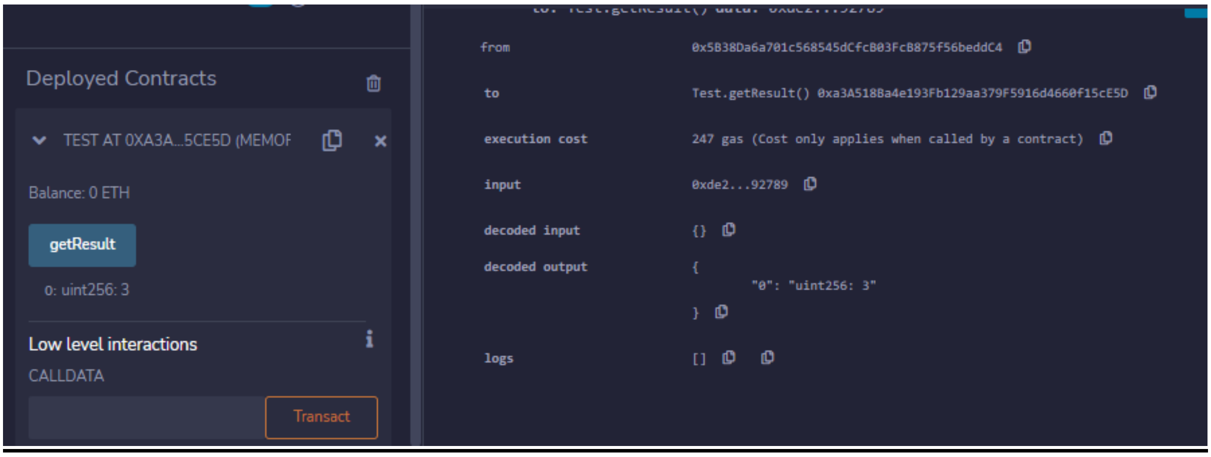
```
from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                B.getResult() 0xF896bB1Da84b8dDE7Ca31D79075B56e51Cdd5582
execution cost    249 gas (Cost only applies when called by a contract)
input             0xde2...92789
decoded input     {}
decoded output    {
                        "0": "uint256: 20100"
                  }
logs              []
>
```

5. 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:



```
to: Test.getResult() data: 0xde2...92789
from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                Test.getResult() 0xa3A518Ba4e193Fb129aa379F5916d4660f15cE5D
execution cost    247 gas (Cost only applies when called by a contract)
input             0xde2...92789
decoded input     {}
decoded output    {
                          "0": "uint256: 3"
                  }
logs              []
```

PRACTICAL 8

Implement and demonstrate the use of the following in Solidity

1. Libraries

Code:

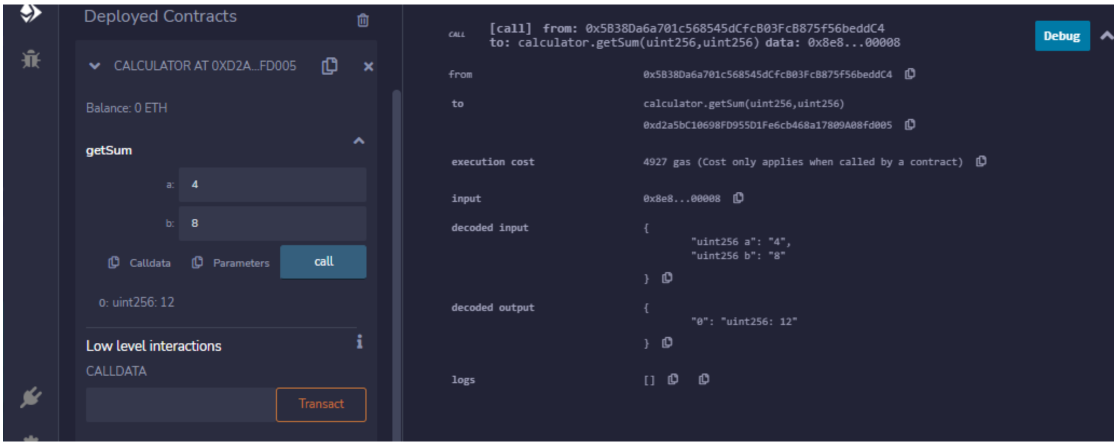
Libraries.sol:

```
pragma solidity ^0.8.0;
import "./MathUtils.sol";
contract calculator{
   using MathUtils for uint;

   function getSum(uint a, uint b) public pure returns(uint){
      return a.add(b);
   }
}
```

MathUtils.sol:

```
pragma solidity ^0.8.0;
library MathUtils{
   function add(uint x, uint y) public pure returns(uint){
      return x+y;
   }
}
```

Output:



```
0: uint256: 12
```

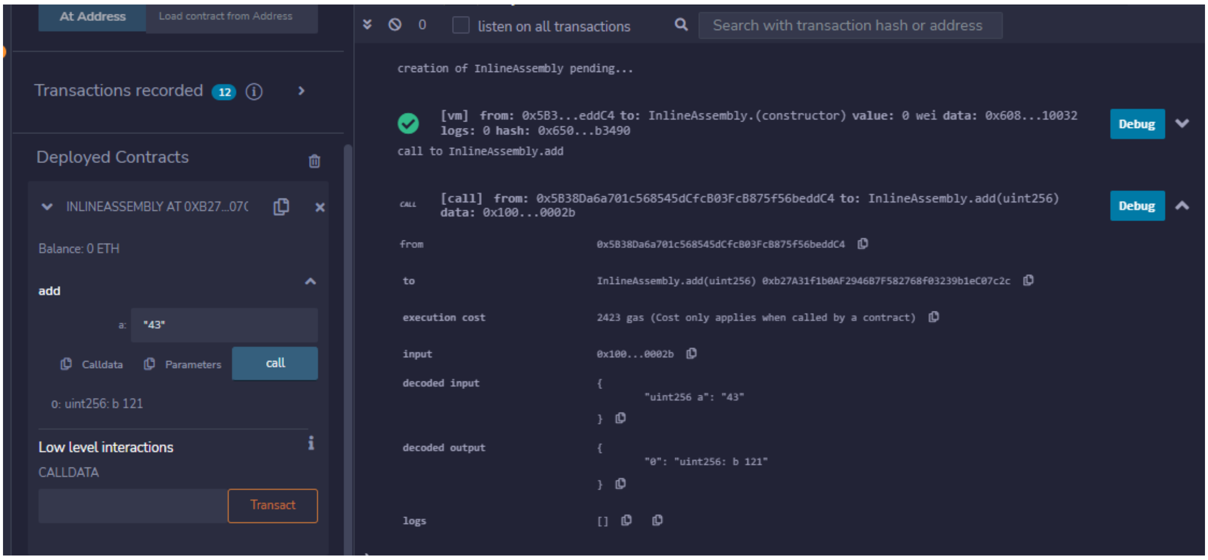
```
[call]  from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to: calculator.getSum(uint256,uint256) data: 0x8e8...00008

from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  calculator.getSum(uint256,uint256)
                    0xd2a5bC10698FD955D1Fe6cb468a17809A08fd005
execution cost      4927 gas (Cost only applies when called by a contract)
input               0x8e8...00008
decoded input       {
                          "uint256 a": "4",
                          "uint256 b": "8"
                    }
decoded output      {
                          "0": "uint256: 12"
                    }
logs                []
```

2. Assembly

Code:

```
pragma solidity ^0.5.0;
contract InlineAssembly {
   function add(uint a) public view returns (uint b) {
      assembly {
         let c := add(a, 56)
         mstore(0x80, c)
         {
            let d := add(sload(c), 22)
            b := d
         }
         b := add(b, c)
      }
   }
}
```

Output:



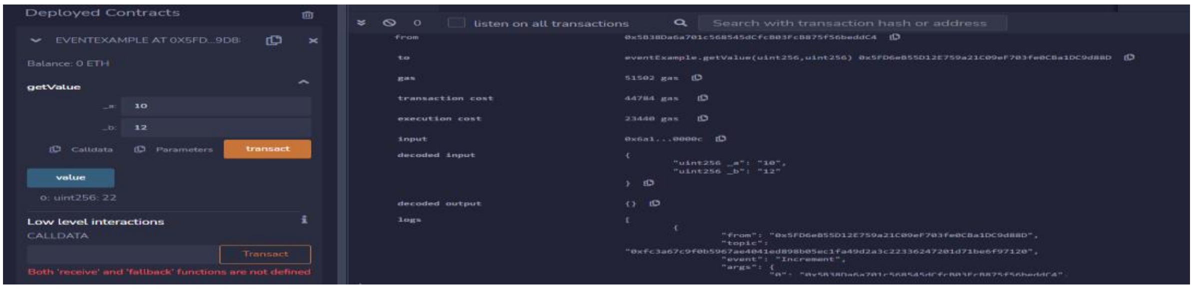
```
creation of InlineAssembly pending...

[vm]  from: 0x5B3...eddC4 to: InlineAssembly.(constructor) value: 0 wei data: 0x608...10032
logs: 0 hash: 0x650...b3490
call to InlineAssembly.add

CALL  [call]  from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: InlineAssembly.add(uint256)
      data: 0x100...0002b
from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                InlineAssembly.add(uint256) 0xb27A31f1b0AF2946B7F582768f03239b1eC07c2c
execution cost    2423 gas (Cost only applies when called by a contract)
input             0x100...0002b
decoded input     {
                         "uint256 a": "43"
                  }
decoded output    {
                         "0": "uint256: b 121"
                  }
logs              []
```

3. Events

Code:

```
pragma solidity ^0.4.21;
contract eventExample {
   uint256 public value = 0;
   event Increment(address owner);
   function getValue(uint _a, uint _b) public { // _a, _b is instance variable (used internally
only)
       emit Increment(msg.sender);
       value = _a + _b;
   }
}
```

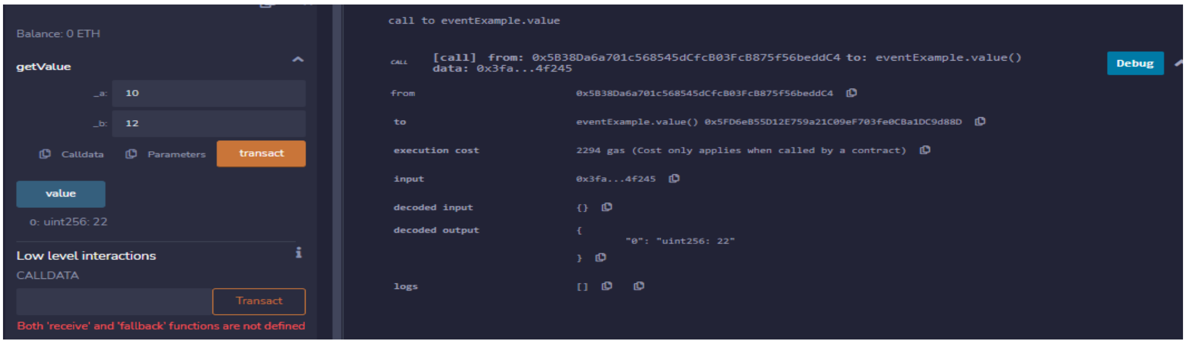
Output:



```
Deployed Contracts
EVENTEXAMPLE AT 0X5FD...9D8!
Balance: 0 ETH
getValue
_a: 10
_b: 12
value
0: uint256: 22
Low level interactions
CALLDATA
Both 'receive' and 'fallback' functions are not defined
```

```
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  eventExample.getValue(uint256,uint256) 0x5FD6eB55D12E759a21C09eF703fe0CBa1DC9d88D
gas                 51502 gas
transaction cost    44784 gas
execution cost      23440 gas
input               0x6a1...0000c
decoded input       {
                            "uint256 _a": "10",
                            "uint256 _b": "12"
                    }
decoded output      {}
logs                [
                            {
                                    "from": "0x5FD6eB55D12E759a21C09eF703fe0CBa1DC9d88D",
                                    "topic":
                    "0xfc3a67c9f0b5967ae4041ed898b05ec1fa49d2a3c22336247201d71be6f97120",
                                    "event": "Increment",
                                    "args": {
                                            "0": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
```

Calldata
Parameters
transact
Transact
listen on all transactions
Search with transaction hash or address



```
Balance: 0 ETH
getValue
_a: 10
_b: 12
value
0: uint256: 22
Low level interactions
CALLDATA
Both 'receive' and 'fallback' functions are not defined
```

```
call to eventExample.value

CALL    [call]  from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: eventExample.value()
        data: 0x3fa...4f245

from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  eventExample.value() 0x5FD6eB55D12E759a21C09eF703fe0CBa1DC9d88D
execution cost      2294 gas (Cost only applies when called by a contract)
input               0x3fa...4f245
decoded input       {}
decoded output      {
                            "0": "uint256: 22"
                    }
logs                []
```

4. Error Handling

a. **Require**
b. **Assert**
c. **Revert**

a. Require:

Code:

```
pragma solidity ^0.5.0;
//RIFATH 3
contract requireStatement {

   function checkInput(uint _input) public view returns(string memory){
      require(_input >= 0, "invalid uint8");
      require(_input <= 255, "invalid uint8");
      return "Input is Uint8";
   }
   function Odd(uint _input) public view returns(bool){
      require(_input % 2 != 0);
      return true;
   }
}
```

Output:

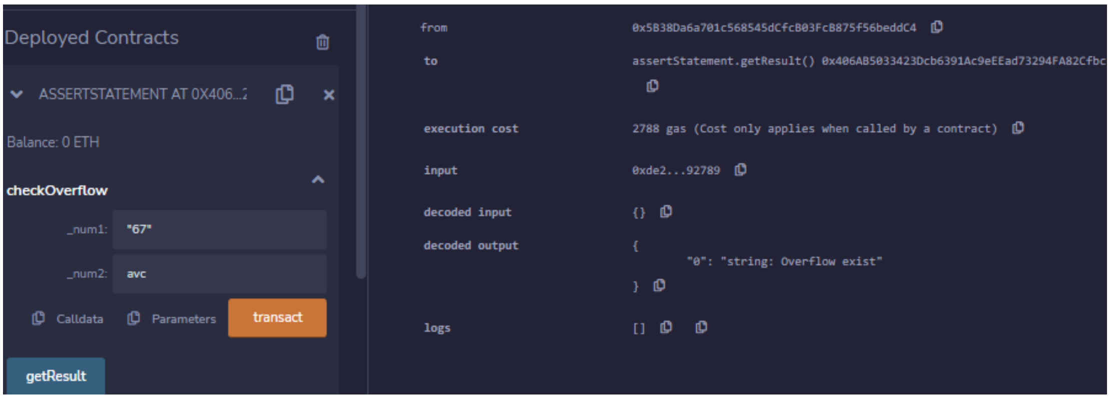
b. Assert:

Code:

```
pragma solidity ^0.5.0;
//RIFATH 3
contract assertStatement {
   bool result;
   function checkOverflow(uint _num1, uint _num2) public {
      uint sum = _num1 + _num2;
      assert(sum<=255);
      result = true;
   }

   function getResult() public view returns(string memory){
      if(result == true){
         return "No Overflow";
      }
      else{
         return "Overflow exist";
      }
   }
}
```

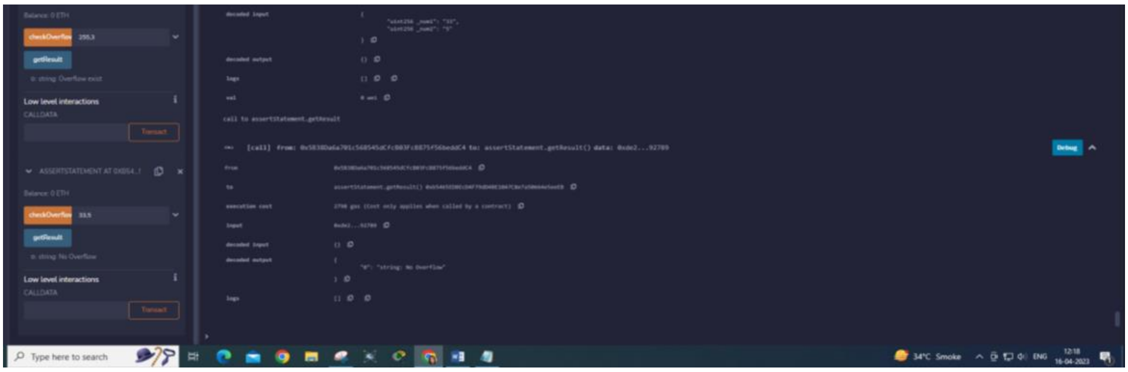
Output:

Overflow exists:



```
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  assertStatement.getResult() 0x406AB5033423Dcb6391Ac9eEEad73294FA82Cfbc
execution cost      2788 gas (Cost only applies when called by a contract)
input               0xde2...92789
decoded input       {}
decoded output      {
                        "0": "string: Overflow exist"
                    }
logs                []
```

No Overflow:



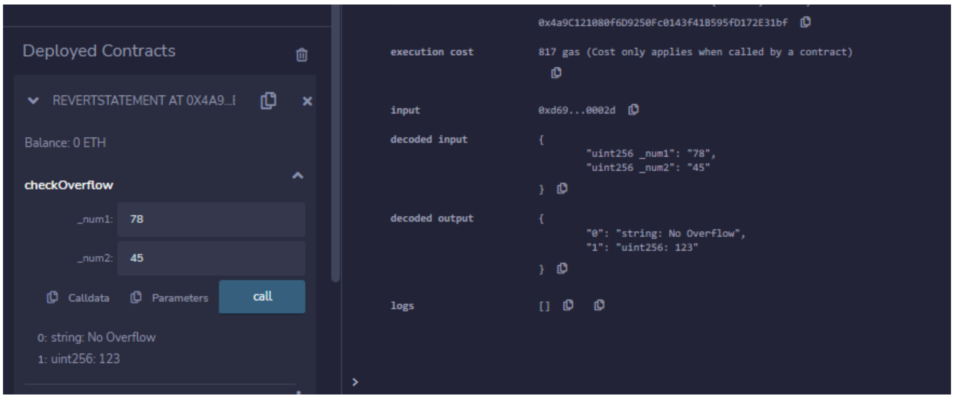
```
call to assertStatement.getResult

[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: assertStatement.getResult() data: 0xde2...92789
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
execution cost      2788 gas (Cost only applies when called by a contract)
input               0xde2...92789
decoded input       {}
decoded output      {
                        "0": "string: No Overflow"
                    }
logs                []
```

c. Revert:

Code:

```
pragma solidity ^0.5.0;
//Rifath 3
contract revertStatement {

function checkOverflow(uint _num1, uint _num2) public view returns(string memory, uint)
{
      uint sum = _num1 + _num2;
      if(sum < 0 || sum > 255){
         revert(" Overflow Exist");
      }
      else{
         return ("No Overflow", sum);
      }
   }
}
```

Output:



Balance: 0 ETH

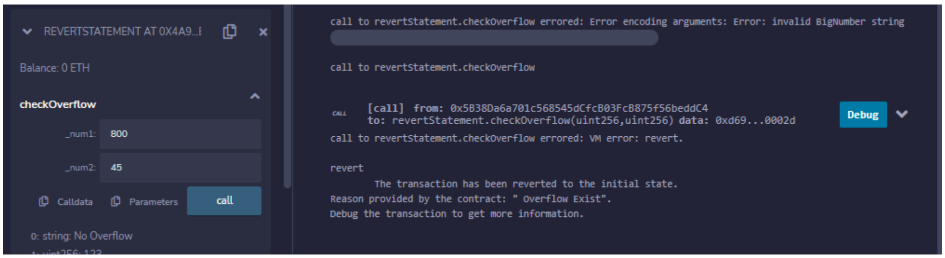
checkOverflow

_num1: 78

_num2: 45

0: string: No Overflow
1: uint256: 123

```
                 0x4a9C121080f6D9250Fc0143f41B595fD172E31bf

execution cost   817 gas (Cost only applies when called by a contract)

input            0xd69...0002d

decoded input    {
                         "uint256 _num1": "78",
                         "uint256 _num2": "45"
                 }

decoded output   {
                         "0": "string: No Overflow",
                         "1": "uint256: 123"
                 }

logs             []
```



Balance: 0 ETH

checkOverflow

_num1: 800

_num2: 45

0: string: No Overflow

```
call to revertStatement.checkOverflow errored: Error encoding arguments: Error: invalid BigNumber string

call to revertStatement.checkOverflow

CALL   [call]  from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
       to: revertStatement.checkOverflow(uint256,uint256) data: 0xd69...0002d
call to revertStatement.checkOverflow errored: VM error: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: " Overflow Exist".
Debug the transaction to get more information.
```