

BLOCKCHAIN



GURU NANAK COLLEGE OF ARTS, SCIENCE & COMMERCE
G.T.B NAGAR, MUMBAI-400037

DEPARTMENT OF INFORMATION TECHNOLOGY

MSc(IT) PART I SEMESTER II

Practical Journal IN
BLOCKCHAIN

Submitted by
NAME- ANIKET HINUKALE
SEAT NO- 2510261

Academic Year
2021-22

BLOCKCHAIN



GURU NANAK COLLEGE OF ARTS, SCIENCE & COMMERCE
G.T.B NAGAR, MUMBAI-400037

DEPARTMENT OF INFORMATION TECHNOLOGY
CERTIFICATE

This is to certify that Mr. / Miss. _____ Aniket HInukale_____

of MSc(IT) Part-I Semester II Seat No. 2510261 has successfully completed the practical's in the subject of **BLOCKCHAIN** as per the requirement of University Of Mumbai in part fulfillment for the completion of Degree of Master of Science (INFORMATION TECHNOLOGY). It is also to certify that this is the original work of the candidate done during the academic year 2021-2022.

Subject In-Charge

In-Charge,MSc(IT)

Date

College Seal

Examiner

Index

Sr.No	Topic	Remark
1	Practical 1 : Write the following programs for Blockchain in Python : (I). A simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it (II). A transaction class to send and receive money and test it .	
2	Practical 2 : Write the following programs for Blockchain in Python : (I).Create multiple transactions and display them . (II). Create a blockchain, a genesis block and execute it .	
3	Practical 3 : Write the following programs for Blockchain in Python : (I).Create a mining function and test it . (II).Add blocks to the miner and dump the blockchain .	
4	Practical 4 : Implement and demonstrate the use of the following in Solidity : (I).Varaible (II).Operators (III).Loops (IV).Decision Making (V).Strings	
5	Practical 5 : Implement and demonstrate the use of the following in Solidity : (I).Arrays (II).Enums (III).Structs (IV).Mappings (V).Coversations (VI).Ether Units (VII).Special Variables	
6	Practical 6: Implement and demonstrate the use of the following in Solidity : (I).Functions	

BLOCKCHAIN

	(II).View Functions	
	(III).Pure Functions	
	(IV).Fallback Functions	
	(V).Function Overloading	
	(VI).Mathematical Functions	
	(VII).Cryptographic Functions	
7	Practical 7 : Implement and demonstrate the use of the following in Solidity :	
	(I).Contracts	
	(II).Inheritance	
	(III).Constructors	
	(IV).Abstract Class	
	(V).Interfaces	
8	Practical 8 : Implement and demonstrate the use of the following in Solidity :	
	(I).Libraries	
	(II).Assembly	
	(III).Events	
	(IV).Error Handling	

Practical 1

Write the following programs for Blockchain in Python :

(I) A simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it

(II) A transaction class to send and receive money and test it .

→

```
# import libraries
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
pip install pycryptodome
# following imports are required by PKI
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

import binascii
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)
```

BLOCKCHAIN

```
@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')

Dinesh = Client()
Ramesh = Client()
t = Transaction(Dinesh, Ramesh.identity, 5.0)
signature = t.sign_transaction()
print(signature)
```

Output:

BLOCKCHAIN

File Edit View Insert Runtime Tools Help Saving...

+ Code + Text

```
[2] # import libraries
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
```

```
[3] pip install pycryptodome
Collecting pycryptodome
  Downloading https://files.pythonhosted.org/packages/ad/16/9627ab0493894a11c68e46000dbcc82f578c8ff06bc2980dc016aea9bd3/pycryptodome-3.10.1-cp35-ab13-manylinux2
  1.9MB 4.8MB/s
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.10.1
```

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[5] import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

import binascii

import binascii
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()
```

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
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')

Dinesh = Client()
Ramesh = Client()

t = Transaction(Dinesh,Ramesh.identity,5.0)

signature = t.sign_transaction()
print (signature)
```

```
656f0e37653d1bc3c538de01595a9c7e4a603e03740d1ed450f1ea32f568663cebfff7ec9757cd35d4f8b1a3524341528a48674981265b68a477c06d57271d6250f45d6e3946a4f8166fff50251956fa
```

Practical 2

Write the following programs for Blockchain in Python :

- (I).Create multiple transactions and display them .
- (II). Create a blockchain, a genesis block and execute it .

→

```
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(
```

ROLL NO 01

ANKIT KUMAR

BLOCKCHAIN

```
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(
```

BLOCKCHAIN

```
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 ('-----')
```

```
class Block:  
    def __init__(self):  
        self.verified_transactions = []
```

BLOCKCHAIN

```
self.previous_block_hash = ""  
self.Nonce = ""  
  
last_block_hash = ""  
  
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 = []  
  
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 ('=====')
```

BLOCKCHAIN

```
TPCoins.append(block0)
```

```
dump_blockchain(TPCoins)
```

Output:

```
[10] 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 ('-----')

[11] transactions = []

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

[13] t1 = Transaction(
    Dinesh,
    Ramesh.identity,
    15.0
)

[15] t1.sign_transaction()
transactions.append(t1)

[16] 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)
```

BLOCKCHAIN

```
[16] 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)

[17] for transaction in transactions:
    display_transaction(transaction)
    print ('-----')

sender: 30819f300d06092a864886f70d010101050003818d0030818
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5
-----
value: 15.0
-----
time: 2021-05-05 07:37:46.751762
-----
sender: 30819f300d06092a864886f70d010101050003818d003081890281810096e3e436b160bf2feecba6d
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5
-----
value: 15.0
-----
time: 2021-05-05 07:37:46.751762
-----
sender: 30819f300d06092a864886f70d010101050003818d003081890281810096e3e436b160bf2feecba6d
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a3b073014e3d9a1e3353
-----
value: 6.0
-----
time: 2021-05-05 07:38:17.042489
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5eb8
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100967bf6965c7e25f7c711
-----
value: 2.0
-----
time: 2021-05-05 07:38:17.044049
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a3b073014e3d9a1e335365f
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5
-----
value: 4.0
-----
time: 2021-05-05 07:38:17.045503
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100967bf6965c7e25f7c711a30
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a3b073014e3d9a1e3353
-----
value: 7.0
-----
time: 2021-05-05 07:38:17.046070

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

[19] last_block_hash = ""
```



BLOCKCHAIN

```
+ Code + Text RAM Disk Editing ^
```

```
[20] Dinesh = Client()

[21] t0 = Transaction (
    "Genesis",
    Dinesh.identity,
    500.0
)

[22] block0 = Block()

[23] block0.previous_block_hash = None
Nonce = None

[24] block0.verified_transactions.append (t0)

[25] digest = hash (block0)
last_block_hash = digest

[26] 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 ('=====')
```

```
[27] TPCoins = []

[28] 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 ('=====')
```

```
[29] TPCoins.append (block0)

[30] dump_blockchain(TPCoins)

Number of blocks in the chain: 1
block # 0
sender: Genesis
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100df7d100622cc76eab161
-----
value: 500.0
-----
time: 2021-05-05 07:41:01.767902
-----
=====
```

Practical 3

Write the following programs for Blockchain in Python :

- (I).Create a mining function and test it .
- (II).Add blocks to the miner and dump the blockchain .

→

```
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
```

```
last_transaction_index = 0
```

```
block = Block()  
for i in range(3):  
    temp_transaction = transactions[last_transaction_index]  
    # validate transaction  
    # if valid  
    block.verified_transactions.append (temp_transaction)  
    last_transaction_index += 1 mine ("test message", 2)
```

```
block.previous_block_hash = last_block_hash  
block.Nonce = mine (block, 2)  
digest = hash (block)  
TPCoins.append (block)  
last_block_hash = digest
```

BLOCKCHAIN

```
# Miner 2 adds a block
block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    # validate transaction
    # if valid
    block.verified_transactions.append (temp_transaction)
    last_transaction_index += 1
    block.previous_block_hash = last_block_hash
    block.Nonce = mine(block, 2)
    digest = hash (block)
    TPCoins.append (block)
    last_block_hash = digest
# Miner 3 adds a block
block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    #display_transaction (temp_transaction)
    # validate transaction
    # if valid
    block.verified_transactions.append (temp_transaction)
    last_transaction_index += 1

    block.previous_block_hash = last_block_hash
    block.Nonce = mine (block, 2)
    digest = hash (block)

    TPCoins.append (block)
    last_block_hash = digest
dump_blockchain(TPCoins)
```

Full Output:

BLOCKCHAIN

```
# import libraries
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

[ ] pip install pycryptodome
Collecting pycryptodome
  Downloading https://files.pythonhosted.org/packages/ad/16/9627ab0493894a11c68e46000dbcc
[ ] | 1.9MB 5.6MB/s
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.10.1

[ ] # following imports are required by PKI
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

[ ] import binascii
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')

[ ]

[ ] Dinesh = Client()
Ramesh = Client()

[ ] t = Transaction(Dinesh,Ramesh.identity,5.0)

[ ] signature = t.sign_transaction()
print (signature)

251f28f6f523733739979611b404c755210b5b6a70f28d5eb3e3049f7dceea873e472f0df41a55152c71bb6f

[ ] def display_transaction(transaction):
    #for transaction in transactions:
    dict = transaction.to_dict()
    print ("sender: " + dict['sender'])
    print ('----')
```

BLOCKCHAIN

```
[ ] 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)
```

BLOCKCHAIN

```
+ Code + Text | ⌂ Copy to Drive Connect ▾ | ⌂ Editing ▾
```

```
[ ] for transaction in transactions:
    display_transaction (transaction)
    print ('-----')

    sender: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4713
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac49
    -----
    value: 15.0
    -----
    time: 2021-04-08 06:10:52.172517
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4713
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 6.0
    -----
    time: 2021-04-08 06:11:40.567785
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac492d1
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e
    -----
    value: 2.0
    -----
    time: 2021-04-08 06:11:40.569278
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86384
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac49
    -----
    value: 4.0
    -----
    time: 2021-04-08 06:11:40.570658
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 7.0
    -----
    time: 2021-04-08 06:11:40.572173
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac492d1
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 3.0
    -----
    time: 2021-04-08 06:11:40.574026

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

[ ] last_block_hash = ""

[ ] Dinesh = Client()

[ ] t0 = Transaction (
    "Genesis",
    Dinesh.identity,
    500.0
)

[ ] block0 = Block()

[ ] block0.previous_block_hash = None
Nonce = None

[ ] block0.verified_transactions.append (t0)
```

BLOCKCHAIN

```
+ Code + Text | Copy to Drive Connect ▾ | ✎ Editing | ^
```

```
[ ] 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)
            print ('-----')
        print ('=====')
```

```
[ ] TPCoins = []

[ ] 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 ('=====')
```

```
[ ] TPCoins.append (block0)

[ ] dump_blockchain(TPCoins)

Number of blocks in the chain: 1
block # 0
sender: Genesis
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100bae5c5a46a1591af94c0
-----
value: 500.0
-----
time: 2021-04-08 06:16:28.464142
-----
=====
```

```
[ ] 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)
'938c72d2197fa6c2294df7bc8cea6be98769aad888e3cfa15558c78469394ebd'

[ ] last_transaction_index = 0

[ ] block = Block()
for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    # validate transaction
    # if valid
    block.verified_transactions.append (temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine (block, 2)
digest = hash (block)
TPCoins.append (block)
last_block_hash = digest
```

BLOCKCHAIN

```
+ Code + Text | ⌂ Copy to Drive Connect ▾ | ✎ Editing | ^
```

```
[ ] # Miner 2 adds a block
block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    # validate transaction
    # if valid
    block.verified_transactions.append (temp_transaction)
    last_transaction_index += 1
block.previous_block_hash = last_block_hash
block.Nonce = mine(block, 2)
digest = hash (block)
TPCoins.append (block)
last_block_hash = digest
# Miner 3 adds a block
block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    #display_transaction (temp_transaction)
    # validate transaction
    # if valid
    block.verified_transactions.append (temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine (block, 2)
digest = hash (block)

TPCoins.append (block)
last_block_hash = digest

[ ] dump_blockchain(TPCoins)
time: 2021-04-08 06:11:40.570658
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
-----
value: 7.0
-----
time: 2021-04-08 06:11:40.572173
-----
sender: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac492d1
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
-----
value: 3.0
-----
time: 2021-04-08 06:11:40.574036
-----
=====
block # 3
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86384
-----
recipient: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4
-----
value: 8.0
-----
time: 2021-04-08 06:11:40.575310
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86384
-----
recipient: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac49
-----
value: 1.0
-----
time: 2021-04-08 06:11:40.576645
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
-----
recipient: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4
-----
value: 5.0
-----
time: 2021-04-08 06:11:40.578007
-----
=====
```

Practical 4

Implement and demonstrate the use of the following in Solidity:

- (I).Varaible
- (II).Operators
- (III).Loops
- (IV).Decision Making
- (V).Strings

(I).Variable

```
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:

BLOCKCHAIN

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with settings for the compiler (version 0.5.17+commit.d19bba13), language (Solidity), EVM version (compiler default), compiler configuration (Auto compile, Enable optimization set to 200), and hide warnings. Below this is a blue button labeled "Compile new one.sol". Under the "CONTRACT" section, it shows "SolidityTest (new one.sol)" with options to "Publish on Swarm" or "Publish on IPFS". At the bottom right, there's a transaction details panel with fields for "from", "to" (SolidityTest.(constructor)), and "gas" (3000000 gas).

```
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
    }
}
```

This screenshot shows the Deploy & Run Transactions interface. It includes fields for "VALUE" (0 wei), "CONTRACT" (SolidityTest - new one.sol), and a prominent orange "Deploy" button. Below the deployment area, there's a "Transactions recorded" section showing a deployed contract at address 0xD91...39138. A "getResult" button is shown under the contract's interface. The right side of the screen is identical to the first screenshot, showing the same transaction details panel.

```
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
    }
}
```

BLOCKCHAIN

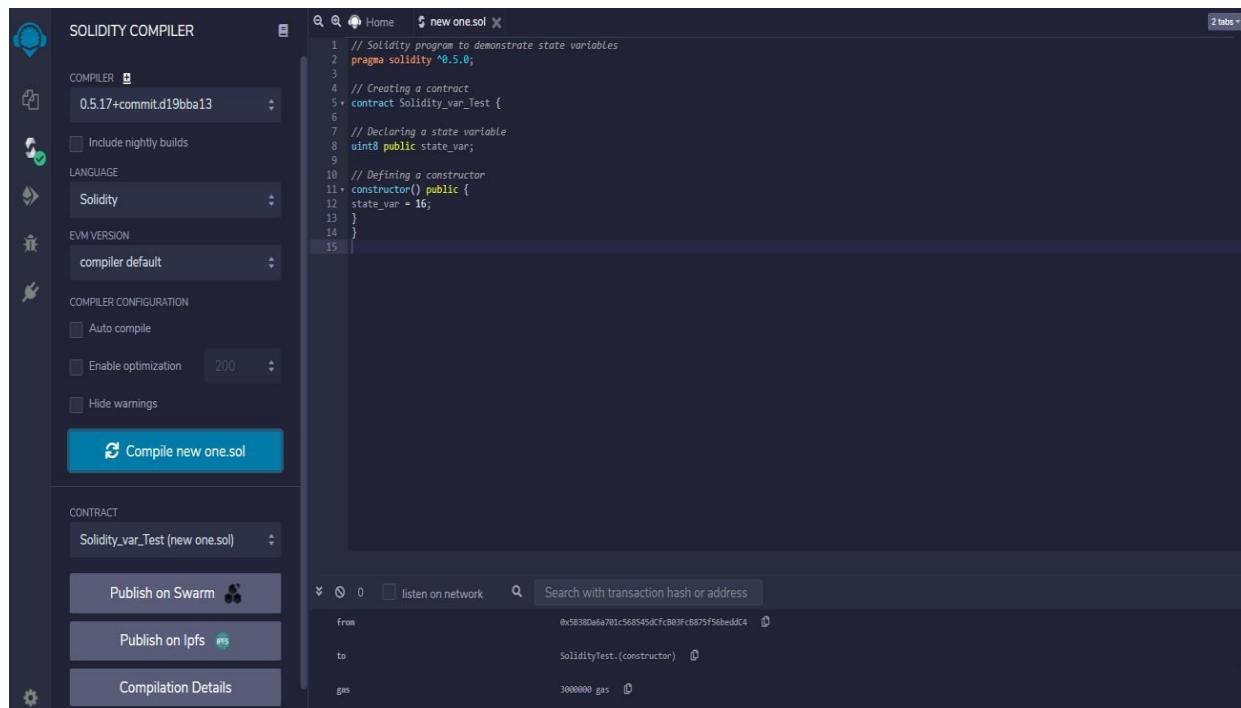
```
// Solidity program to demonstrate state variables
pragma solidity ^0.5.0;

// Creating a contract
contract Solidity_var_Test {

// Declaring a state variable
uint8 public state_var;

// Defining a constructor
constructor() public {
state_var = 16;
}
}
```

Output:



The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with compiler settings like version (0.5.17+commit.d196ba13), language (Solidity), EVM version (compiler default), and compiler configuration options (Auto compile, Enable optimization set to 200, Hide warnings). A prominent blue button at the bottom of this sidebar says "Compile new one.sol". Below the sidebar, a "CONTRACT" section shows "Solidity_var_Test (new one.sol)". Underneath are buttons for "Publish on Swarm" and "Publish on Ipfs". At the very bottom is a "Compilation Details" button. The main workspace on the right displays the Solidity code. Below the code, a transaction details section shows a transaction from address 0x56380a6a701c568545dCf:883Fc8873f56be0dC4 to the contract address SolidityTest.(constructor) with a gas limit of 3000000.

```
// Solidity program to demonstrate state variables
pragma solidity ^0.5.0;

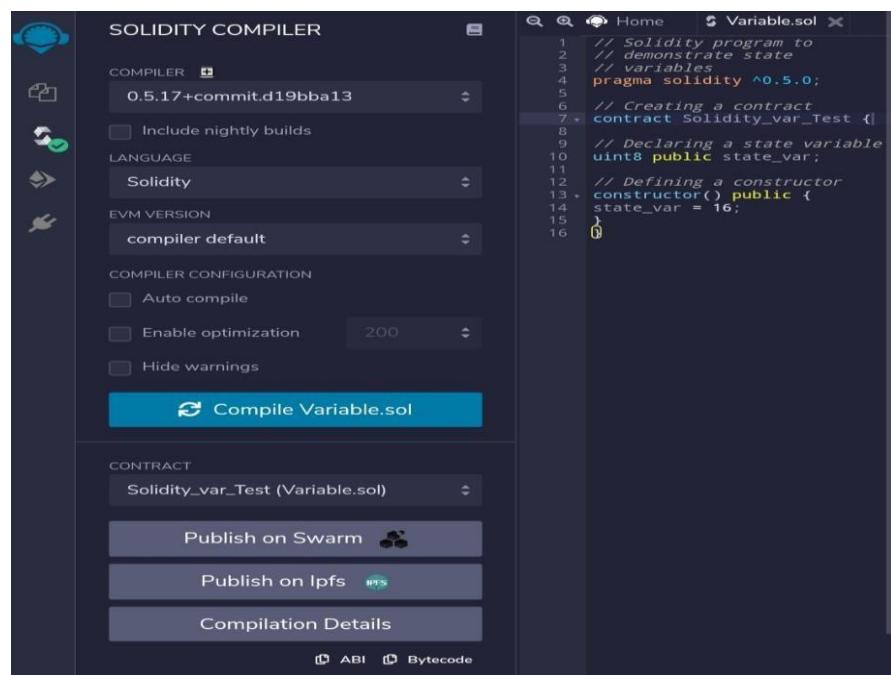
// Creating a contract
contract Solidity_var_Test {

// Declaring a state variable
uint8 public state_var;

// Defining a constructor
constructor() public {
state_var = 16;
}
}

0x56380a6a701c568545dCf:883Fc8873f56be0dC4
SolidityTest.(constructor)
3000000 gas
```

BLOCKCHAIN



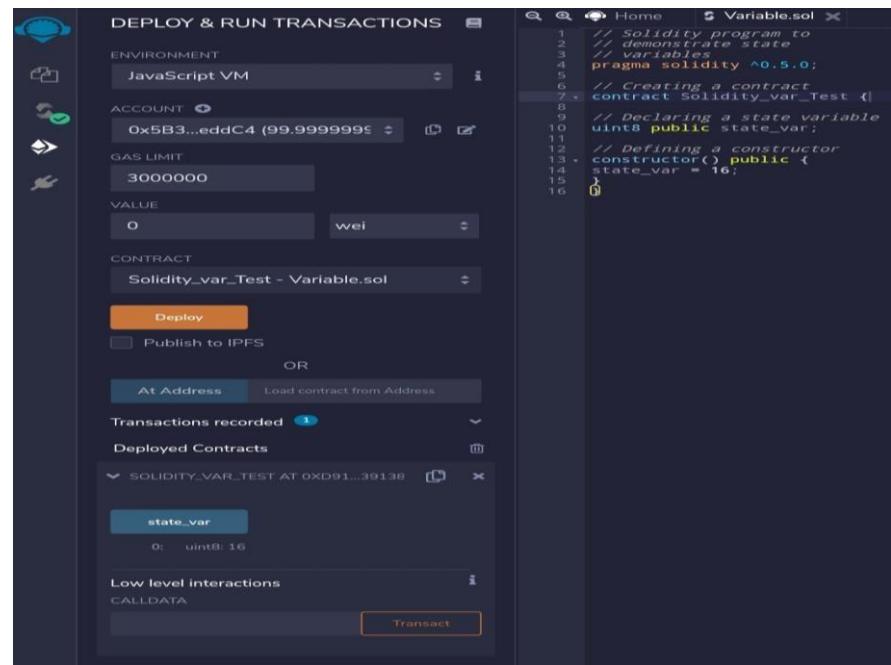
The Solidity Compiler interface shows the compilation of a Solidity program named `Variable.sol`. The code defines a contract `Solidity_var_Test` with a state variable `state_var` initialized to 16.

```
// Solidity program to demonstrate state variables
pragma solidity ^0.5.0;

// Creating a contract
contract Solidity_var_Test {
    // Declaring a state variable
    uint8 public state_var;
}

// Defining a constructor
constructor() public {
    state_var = 16;
}
```

The interface includes sections for Compiler Configuration (Auto compile, Enable optimization, Hide warnings), Contract Selection (Solidity_var_Test (Variable.sol)), and Publishing options (Publish on Swarm, Publish on lpfs). ABI and Bytecode links are also provided.



The Deploy & Run Transactions interface shows the deployment of the previously compiled contract `Solidity_var_Test`. The deployment environment is set to JavaScript VM, with account `0x5B3...eddC4` and gas limit `3000000`. The deployed contract address is `SOLIDITY_VAR_TEST AT 0xD91...39138`, and its state variable `state_var` is set to 0.

```
// Solidity program to demonstrate state variables
pragma solidity ^0.5.0;

// Creating a contract
contract Solidity_var_Test {
    // Declaring a state variable
    uint8 public state_var;
}

// Defining a constructor
constructor() public {
    state_var = 16;
}
```

The interface also displays the deployed contracts section, showing the deployed contract and its state variable value.

BLOCKCHAIN

```
// Solidity program to show Global variables
```

```
pragma solidity ^0.5.0;
```

```
// Creating a contract
```

```
contract Test {
```

```
// Defining a variable
```

```
address public admin;
```

```
// Creating a constructor to
```

```
// use Global variable
```

```
constructor() public {
```

```
admin = msg.sender;
```

```
}
```

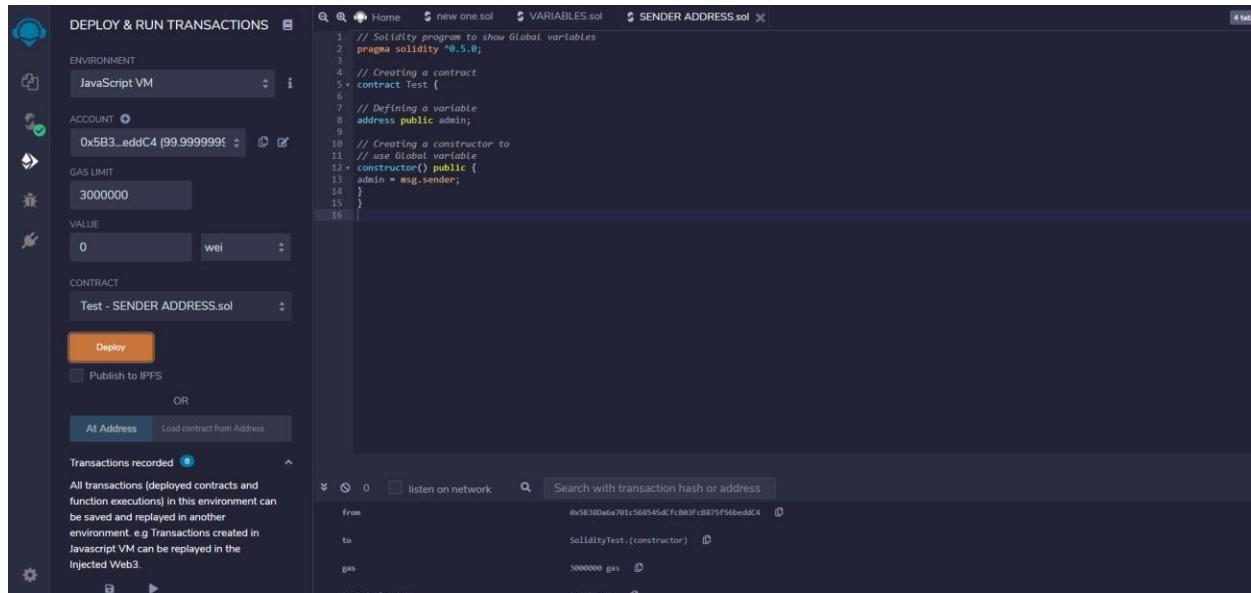
```
}
```

Output:

The screenshot shows the Truffle UI interface. On the left, the 'SOLIDITY COMPILER' sidebar is open, displaying compiler settings: version 0.5.17+commit.d19bba13, language Solidity, EVM version compiler default, and compiler configuration options like auto compile and enable optimization (set to 200). A prominent blue button at the bottom of this sidebar says 'Compile SENDER ADDRESS.sol'. Below the sidebar, the 'CONTRACT' section shows 'Test (SENDER ADDRESS.sol)' selected. To the right, the main workspace displays the Solidity code for 'SENDER ADDRESS.sol'. The code defines a contract 'Test' with a public variable 'admin' and a constructor that sets it to the message sender. The right side also shows a transaction details panel with fields for 'From' (0x58380de781c568545dCfc083fcb875f56bed8C4), 'to' (SolidityTest.(constructor)), and 'gas' (3800000 gas).

```
1 // Solidity program to show Global variables
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5 contract Test {
6
7 // Defining a variable
8 address public admin;
9
10 // Creating a constructor to
11 // use Global variable
12 constructor() public {
13 admin = msg.sender;
14 }
15 }
16
```

BLOCKCHAIN



(II).Operators

```
// Solidity contract to demonstrate Arithmetic Operator
```

```
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;
```

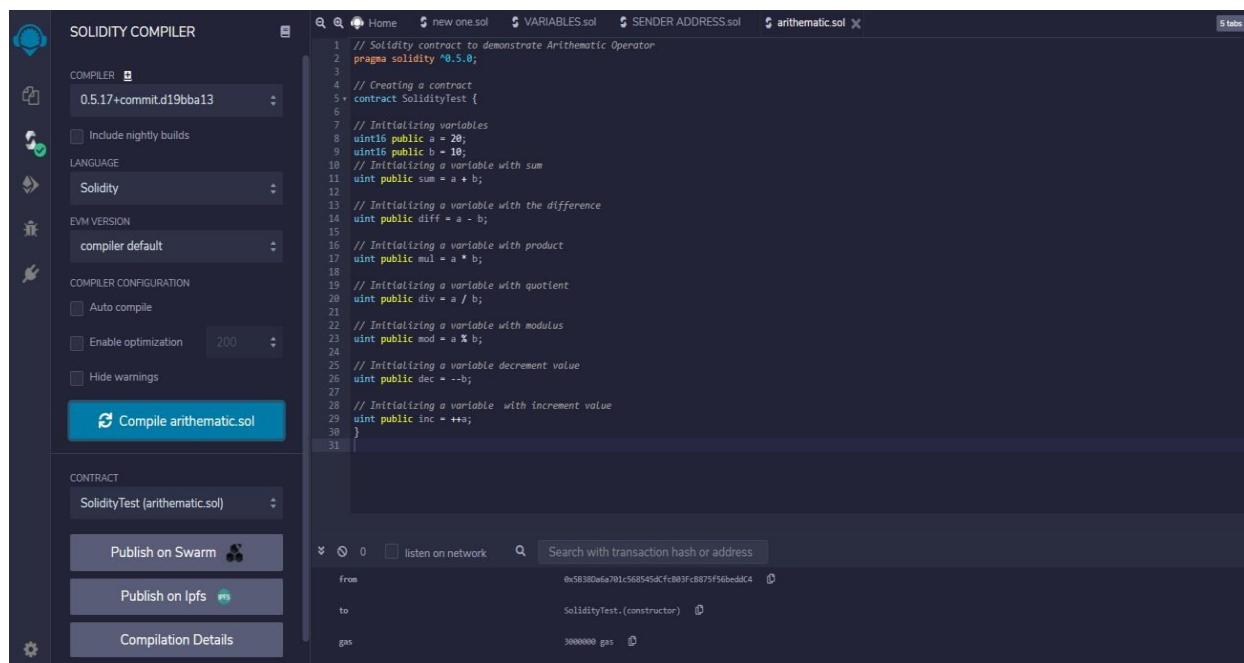
```
// Initializing a variable with quotient
```

```
uint public div = a / b;
```

BLOCKCHAIN

```
// 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:



The screenshot shows the Solidity Compiler interface. On the left, the compiler configuration is set to version 0.5.17+commit.d19bba13, language Solidity, and EVM version compiler default. Optimization is enabled at 200. The right side displays the Solidity code for `arithmetic.sol`. The code initializes variables `a` and `b`, calculates sum, difference, product, quotient, modulus, and decrements/increments `dec` and `inc`. Below the code, the contract `SolidityTest` is selected. The deployment section shows the transaction details: from address 0x5B38Dafa701c568545dCfc883FcB875f56beddC4, to address SolidityTest.(constructor), and gas limit 3000000.

```
// Solidity contract to demonstrate Arithmetic Operator  
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;  
}
```

BLOCKCHAIN

The screenshot shows the Truffle UI interface for deploying and running Solidity contracts. The main area displays the Solidity code for a simple arithmetic test contract:

```
// Solidity contract to demonstrate Arithmetic Operator
pragma solidity ^0.5.0;

// Creating a contract
contract SolidityTest {

    // Initializing variables
    uint8 public a = 20;
    uint8 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 = +b;
}
```

The interface includes sections for ENVIRONMENT (JavaScript VM), ACCOUNT (0x5B3...eddC4), GAS LIMIT (3000000), VALUE (0 wei), CONTRACT (SolidityTest - arithmetic.sol), and a Deploy button. Below the contract details, there are options to Publish to IPFS or At Address. The bottom section shows Transactions recorded with 9 items, and a search bar for transaction hash or address.

The screenshot shows the Truffle UI interface for deploying and running transactions. The top navigation bar includes tabs for Home, new one sol, VARIABLES.sol, SENDER ADDRESS.sol, and arithmetic.sol. The main workspace displays the Solidity source code for the arithmetic contract, which initializes variables a and b, and performs operations like sum, diff, mul, div, inc, mod, and dec.

```
// Solidity contract to demonstrate Arithmetic Operator
pragma solidity ^0.5.0;

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;
}
```

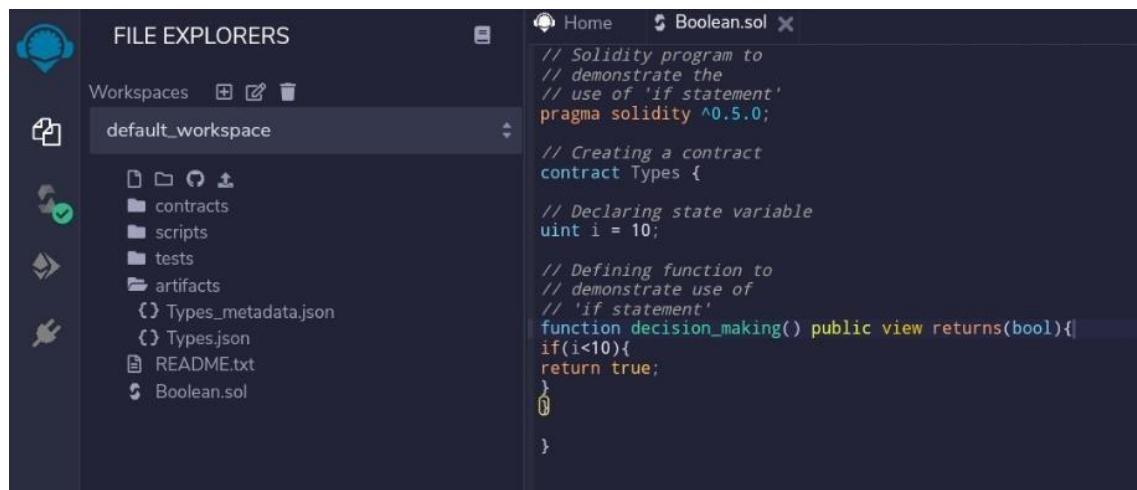
On the left side, there's a sidebar titled "DEPLOY & RUN TRANSACTIONS" with icons for network status, deployment, and running. Below it, a "SOLIDITYTEST AT 0xD2A...FD005 (MEM)" tab is open, showing buttons for each arithmetic operation: a, b, dec, diff, div, inc, mod, mul, and sum. At the bottom, there are sections for "Low level interactions" and "CALLDATA", along with a "Transact" button.

BLOCKCHAIN

(IV).Decision Making

```
// Solidity program to demonstrate the use of 'if statement'  
pragma solidity ^0.5.0;  
  
// Creating a contract  
contract Types {  
    // Declaring state variable  
    uint i = 10;  
  
    // Defining function to demonstrate use of 'if statement'  
    function decision_making() public view returns(bool){  
        if(i<10){  
            return true;  
        }  
    }  
}
```

Output:



The screenshot shows the Solidity IDE interface. On the left is the 'FILE EXPLORERS' sidebar, which displays the project structure under 'default_workspace'. It includes folders for contracts, scripts, tests, and artifacts, along with JSON files like 'Types_metadata.json' and 'Types.json', and text files 'README.txt' and 'Boolean.sol'. The main workspace on the right shows the Solidity code for 'Boolean.sol'.

```
// Solidity program to  
// demonstrate the  
// use of 'if statement'  
pragma solidity ^0.5.0;  
  
// Creating a contract  
contract Types {  
    // Declaring state variable  
    uint i = 10;  
  
    // Defining function to  
    // demonstrate use of  
    // 'if statement'  
    function decision_making() public view returns(bool){  
        if(i<10){  
            return true;  
        }  
    }  
}
```

BLOCKCHAIN

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons (headphones, file, settings, etc.). The main area has tabs for Home and Boolean.sol. The Boolean.sol tab is active, displaying the following Solidity code:

```
// Solidity program to
// demonstrate the
// use of 'if statement'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variable
    uint i = 10;

    // Defining function to
    // demonstrate use of
    // 'if statement'
    function decision_making() public view returns(bool){
        if(i<10){
            return true;
        }
    }
}
```

Below the code, there's a large blue button labeled "Compile Boolean.sol". Under the CONTRACT section, it shows "Types (Boolean.sol)". There are three buttons for publishing: "Publish on Swarm" (with a Swarm icon), "Publish on Ipfs" (with an IPFS icon), and "Compilation Details". At the bottom, there are links for ABI and Bytecode.

The screenshot shows the Truffle UI interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar includes sections for ENVIRONMENT (JavaScript VM), ACCOUNT (0x5B3...eddC4), GAS LIMIT (3000000), and VALUE (0 wei). Below these are CONTRACT and DEPLOYMENT sections. The CONTRACT section shows 'Types - Boolean.sol'. The DEPLOYMENT section has a 'Deploy' button and a checked 'Publish to IPFS' checkbox. An 'OR' section allows selecting 'At Address' or 'Load contract from Address'. The 'Transactions recorded' section shows one entry: 'Deployed Contracts' with 'TYPES AT 0xD91...39138 (MEMORY)'. This entry is expanded to show a 'decision_making...' button, which is highlighted in blue, indicating it is currently selected. Below this, it says '0: bool: false'. At the bottom, there's a 'Low level interactions' section with a 'CALLDATA' tab and a 'Transact' button.

Home Boolean.sol

```
// Solidity program to
// demonstrate the
// use of 'if statement'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variable
    uint i = 10;

    // Defining function to
    // demonstrate use of
    // 'if statement'
    function decision_making() public view returns(bool){
        if(i<10){
            return true;
        }
    }
}
```

BLOCKCHAIN

```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variables
    uint i = 10;
    bool even;

    // Defining function to
    // demonstrate the use of
    // 'if...else statement'
    function decision_making() public {
        if(i%2 == 0){
            even = true;
        }
        else{
            even = false;
        }
    }

    function getresult() public view returns(bool)
    {
        return even;
    }
}
```

Output:

BLOCKCHAIN

FILE EXPLORERS

Workspaces default_workspace

- contracts
- scripts
- tests
- artifacts
- Types_metadata.json
- Types.json
- test_metadata.json
- test.json
- README.txt
- Boolean.sol
- Book.sol
- If_else.sol

```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variables
    uint i = 10;
    bool even;

    // Defining function to demonstrate the use of 'if...else statement'
    function decision_making() public {
        if(i%2 == 0){
            even = true;
        }
        else{
            even = false;
        }
    }

    function getResult() public view returns(bool) {
        return even;
    }
}
```

SOLIDITY COMPILER

COMPILER 0.5.17+commit.d19bba13

LANGUAGE Solidity

EVM VERSION compiler default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

Compile If_else.sol

CONTRACT Types (If_else.sol)

Publish on Swarm

Publish on IPFS

Compilation Details

ABI Bytecode

```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variables
    uint i = 10;
    bool even;

    // Defining function to demonstrate the use of 'if...else statement'
    function decision_making() public {
        if(i%2 == 0){
            even = true;
        }
        else{
            even = false;
        }
    }

    function getResult() public view returns(bool) {
        return even;
    }
}
```

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT JavaScript VM

ACCOUNT 0x5B3...eddC4 (99.9999995)

GAS LIMIT 3000000

VALUE 0 wei

CONTRACT Types - If_else.sol

Deploy Publish to IPFS

At Address Load contract from Address

Transactions recorded Deployed Contracts

TYPES AT 0xD91...38138 (MEMORY)

TEST AT 0xDB8...33FA8 (MEMORY)

TYPES AT 0xDA0...42B53 (MEMORY)

decision_making

getresult

0: book: true

Low level interactions CALLDATA

Transact

```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variables
    uint i = 10;
    bool even;

    // Defining function to demonstrate the use of 'if...else statement'
    function decision_making() public {
        if(i%2 == 0){
            even = true;
        }
        else{
            even = false;
        }
    }

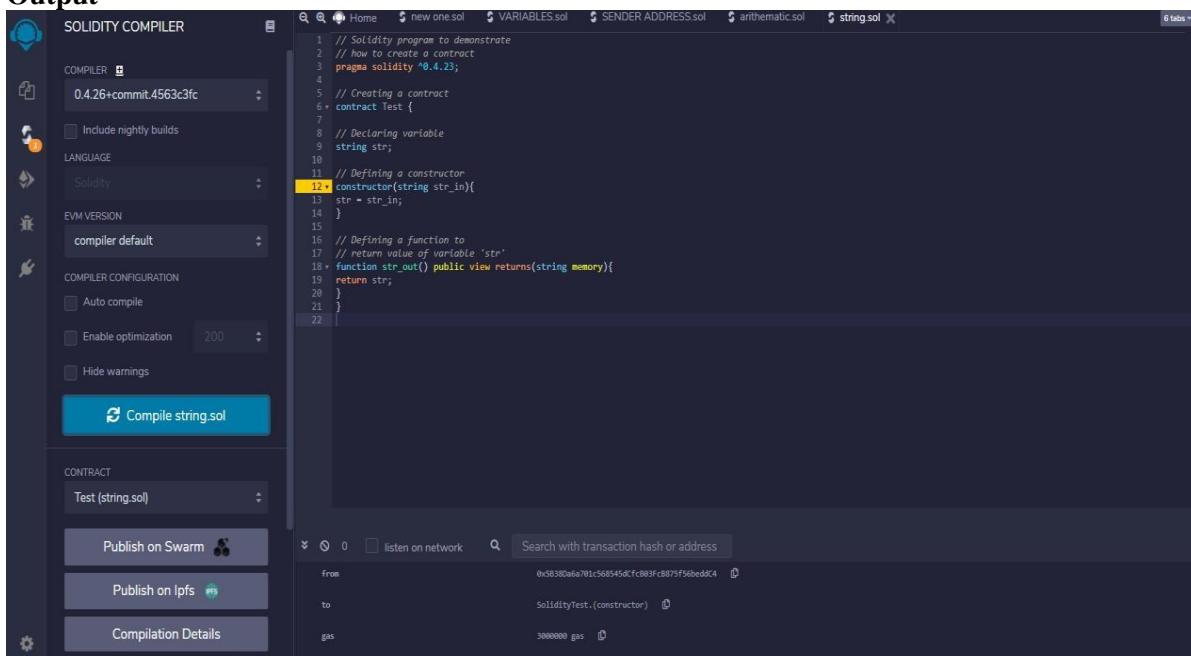
    function getResult() public view returns(bool) {
        return even;
    }
}
```

BLOCKCHAIN

(III) Strings

```
// Solidity program to demonstrate  
// how to create a contract  
pragma solidity ^0.4.23;  
  
// Creating a contract  
contract Test {  
  
    // Declaring variable  
    string str;  
  
    // Defining a constructor  
    constructor(string str_in){  
        str = str_in;  
    }  
  
    // Defining a function to  
    // return value of variable 'str'  
    function str_out() public view returns(string memory){  
        return str;  
    }  
}
```

Output



The screenshot shows the Solidity Compiler interface with the following details:

- SOLIDITY COMPILER** tab is selected.
- COMPILER**: 0.4.26+commit.4563c3fc
- LANGUAGE**: Solidity
- EVM VERSION**: compiler default
- COMPILE CONFIGURATION**: Auto compile, Enable optimization (gas limit: 200), Hide warnings
- Compile button**: Compile string.sol
- CONTRACT**: Test (string.sol)
- Publish options**: Publish on Swarm, Publish on IPFS
- Compilation Details** button
- Output Area**: Displays the input Solidity code with line numbers 1 through 22. Line 12, which contains the constructor definition, is highlighted in yellow.
- Transaction Details**: Shows a transaction with hash 0xd830a6a701c56845dcfc883fc807f56bedd4, from address 0xd830a6a701c56845dcfc883fc807f56bedd4, to address SolidityTest.(constructor), and gas limit 3000000.

BLOCKCHAIN

The screenshot shows a blockchain development environment with the following components:

- Top Bar:** Home, new onesol, VARIABLES.sol, SENDER ADDRESS.sol, arithmetic.sol, string.sol.
- Left Sidebar:** DEPLOY & RUN TRANSACTIONS, showing a list of contracts:
 - SOLIDITY_VAR_TEST at 0x958...D5EE3 (state_var)
 - TEST AT 0X9D7...B5E99 (MEMORY)
 - SOLIDITYTEST at 0xD2A...FD005 (MEMORY)
 - TEST AT 0XDDA...54B2D (MEMORY)
- Middle Panel:** A code editor window displaying the following Solidity code:

```
1 // Solidity program to demonstrate
2 // how to create a contract
3 pragma solidity ^0.4.23;
4
5 // Creating a contract
6+ contract Test {
7
8 // Declaring variable
9     string str;
10
11 // Defining a constructor
12+ constructor(string str_in){
13     str = str_in;
14 }
15
16 // Defining a function to
17 // return value of variable 'str'
18+ function str_out() public view returns(string memory){
19     return str;
20 }
21
22 }
```
- Bottom Panel:** Transaction details and search bar.
 - From: 0x5B3BD6a701c56d545dCfC803Fc8B7f56beidd4
 - To: SolidityTest.(constructor)
 - Gas: 3000000 gas

Search with transaction hash or address:

Practical 5

Implement and demonstrate the use of the following in Solidity:

- (I).Arrays
- (II).Enums
- (III).Structs
- (IV).Mappings
- (V).Coversations
- (VI).Ether Units
- (VII).Special Varaibles

(I).Arrays

```
// Solidity program to demonstrate
// creating a fixed-size array
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variables
    // of type array
    uint[6] data1;
    int[5] data;

    // Defining function to add
    // values to an array
    function array_example() public returns (int[5] memory, uint[6] memory){
        data = [int(50), -63, 77, -28, 90];
        data1 = [uint(10), 20, 30, 40, 50, 60];
    }

    function getresult() public view returns (int[5] memory,uint[6] memory){
        return (data, data1);
    }
}
```

BLOCKCHAIN

Output:

The screenshot shows the Truffle UI interface divided into two main sections: the left sidebar and the right main area.

Left Sidebar (Solidity Compiler):

- SOLIDITY COMPILER** section:
 - Compiler: 0.5.17+commit.d19bba13
 - Include nightly builds:
 - Language: Solidity
 - EVM Version: compiler default
 - Compiler Configuration:
 - Auto compile:
 - Enable optimization: 200
 - Hide warnings:
 - Compile button: **Compile Array.sol**

Right Main Area:

 - FILE EXPLORERS** section:
 - Workspaces: default_workspace
 - Files listed: contracts, scripts, tests, artifacts, README.txt, Variable.sol, Sender Address.sol, String.sol, **Array.sol**
 - Code Editor:** A tab titled "Array.sol" containing the following Solidity code:

```
1 // Solidity program to demonstrate
2 // creating a fixed-size array
3 pragma solidity ^0.5.0;
4
5 // Creating a contract
6 contract Types {
7
8     // Declaring state variables
9     // of type array
10    uint[6] data1;
11    int[5] data;
12
13    // Defining function to add
14    // values to an array
15    function array_example() public returns (int[5] memory, uint[6] memory){
16        data = [int(50), -63, 77, -28, 90];
17        data1 = [uint(10), 20, 30, 40, 50, 60];
18    }
19
20    function getresult() public view returns (int[5] memory,uint[6] memory){
21
22        return (data, data1);
23    }
24 }
```

BLOCKCHAIN

The screenshot shows a blockchain development interface with the following details:

- Environment:** JavaScript VM
- Account:** 0x5B3...eddC4 (99.99999999999999)
- Gas Limit:** 3000000
- Value:** 0 wei
- Contract:** Types - Array.sol
- Deploy:** Orange button
- Publish to IPFS:** Checkbox
- OR**
- At Address:** Load contract from Address
- Transactions recorded:** 3
- Deployed Contracts:** TYPES AT 0xD91...39138 (MEMORY)
- array_example:** orange button
- getresult:** blue button
- Low level interactions:** CALLDATA
- Transact:** Orange button

```
// Solidity program to demonstrate
// creating a fixed-size array
pragma solidity ^0.5.0;

// Creating a contract
contract Types {
    // Declaring state variables
    // of type array
    uint[6] data1;
    int[5] data;

    // Defining function to add
    // values to an array
    function array_example() public returns (int[5] memory, uint[6] memory){
        data = [int(50), -63, 77, -28, 90];
        data1 = [uint(10), 20, 30, 40, 50, 60];
    }
    function getResult() public view returns (int[5] memory, uint[6] memory){
        return (data, data1);
    }
}
```

BLOCKCHAIN

(III).Structs

```
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:

The screenshot shows the Solidity Compiler interface. On the left, the 'SOLIDITY COMPILER' sidebar includes settings for the compiler version (0.5.17+commit.d19bba13), language (Solidity), EVM version (compiler default), and compiler configuration (Auto compile, Enable optimization at 200, Hide warnings). A prominent blue button at the bottom of this sidebar says 'Compile books.sol'. The main workspace displays the Solidity code for the 'test' contract. Below the code, the 'CONTRACT' dropdown shows 'test (books.sol)'. At the bottom of the interface, there are buttons for 'Publish on Swarm' and 'Publish on Ipfs', along with a 'Compilation Details' section. The bottom right corner shows a transaction details panel with fields for 'from', 'to', and 'gas', and a search bar for transaction hashes.

```
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;
    }
}
```

BLOCKCHAIN

```
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;
    }
}
```

TEST AT 0x0FC_9A836 (MEMORY)

setBook

getBookId

0: uint256: 1

Low level interactions

CALLDATA

Transact

From: 0x5880a6a701c568545d7fc883fc8875f56beddc4

To: SolidityTest.(constructor)

Gas: 3000000 gas

transaction cost: 110859 gas

Search with transaction hash or address:

(IV).Mappings

```
pragma solidity ^0.5.0;

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

Output:

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

Include nightly builds

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION

Auto compile

Enable optimization: 200

Hide warnings

Compile Mapping.sol

CONTRACT: LedgerBalance (Mapping.sol)

Publish on Swarm

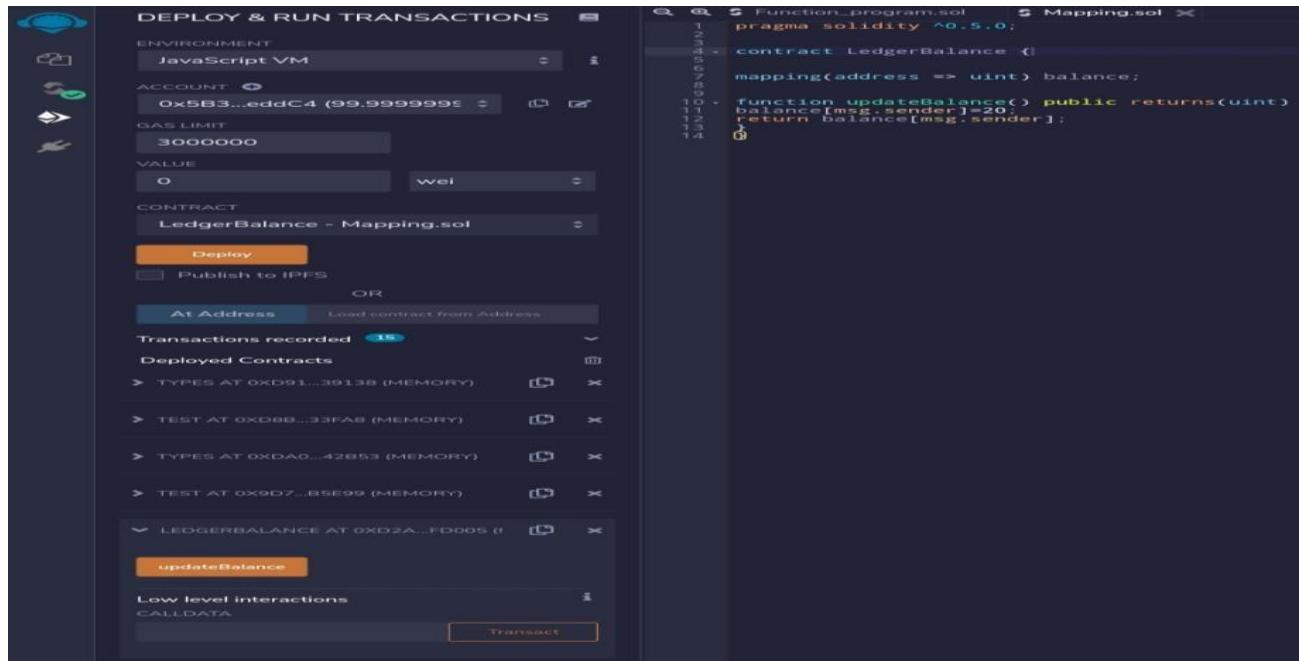
Publish on Ipfs

Compilation Details

ABI Bytecode

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

BLOCKCHAIN



```
pragma solidity ^0.5.0;
contract LedgerBalance {
    mapping(address => string) name;

    function updateBalance() public returns(string memory){
        name[msg.sender] = "Dip";
        return name[msg.sender];
    }

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

Output:

BLOCKCHAIN

The screenshot shows a blockchain development interface. On the left, a "FILE EXPLORERS" panel displays a "default_workspace" folder containing several Solidity files: Boolean.sol, Book.sol, If_else.sol, Function_program.sol, Mapping.sol, and Mapping_string.sol. The "Mapping_string.sol" file is currently selected. On the right, a code editor window shows the Solidity source code for the "LedgerBalance" contract:

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

The screenshot shows the "SOLIDITY COMPILER" interface. It includes settings for the compiler version (0.5.17+commit.d19bba13), language (Solidity), and EVM version (compiler default). Under "COMPILER CONFIGURATION", there are options for auto-compile, enable optimization (set to 200), and hide warnings. A prominent blue button labeled "Compile Mapping_string.sol" is visible. Below the compiler section, a "CONTRACT" dropdown is set to "LedgerBalance (Mapping_string.sol)". There are three buttons for publishing: "Publish on Swarm" (with a Swarm icon), "Publish on Ipfs" (with an IPFS icon), and "Compilation Details". At the bottom, there are links for ABI and Bytecode.

The screenshot shows a "TEST AT 0xD8B...33FA8 (MEMORY)" interface. It lists several memory locations and their values. The "LEDGERBALANCE AT 0xD2A...FD005" entry is expanded, showing two buttons: "updateBalance" and "printsender". Below this, it shows the address: 0x5B38Da6a701c568545dCfcB 03FcB875f56beddC4. A "Low level interactions" section includes a "CALldata" input field and a "Transact" button.

Practical 6

Implement and demonstrate the use 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

(I).Functions

```
pragma solidity ^0.5.0;
contract SolidityTest {
    function testpgmresult() public view returns(uint){
        uint a = 1000; // local variable
        uint b = 2000;
        uint result = a + b;
        return result; //access the state variable
    }
}
```

Output:

The screenshot shows the Solidity Compiler interface. On the left, the 'SOLIDITY COMPILER' sidebar includes settings for the compiler version (0.5.17+commit.d19bba13), language (Solidity), EVM version (compiler default), and compiler configuration (Auto compile, Enable optimization set to 200, Hide warnings). A prominent blue button at the bottom of this sidebar says 'Compile Function test.sol'. Below this, the 'CONTRACT' section shows 'SolidityTest (Function test.sol)' and three buttons: 'Publish on Swarm' (with a Swarm icon), 'Publish on Ipfs' (with an IPFS icon), and 'Compilation Details'. On the right, the main workspace displays the Solidity code for 'testpgmresult()' and its corresponding assembly output. The assembly code is shown in a dark-themed text editor.

```
pragma solidity ^0.5.0;
contract SolidityTest {
    function testpgmresult() public view returns(uint){
        uint a = 1000; // local variable
        uint b = 2000;
        uint result = a + b;
        return result; //access the state variable
    }
}
```

BLOCKCHAIN

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for file operations. The main area has tabs for 'Home', 'Function test.sol', and 'Function getresult.sol'. The 'Function test.sol' tab is active, displaying the following Solidity code:

```
1 pragma solidity ^0.5.0;
2 contract SolidityTest {
3     Function testpgmresult() public view returns(uint){
4         uint a = 1000; // local variable
5         uint b = 2000;
6         uint result = a + b;
7         return result; //access the state variable
8     }
9 }
10 }
```

The UI includes fields for 'ENVIRONMENT' (JavaScript VM), 'ACCOUNT' (0x5B3...eddC4), 'GAS LIMIT' (3000000), and 'VALUE' (0 wei). Below these, there's a 'CONTRACT' section for 'SolidityTest - Function test.sol' with a 'Deploy' button. There are also sections for 'Transactions recorded' and 'Deployed Contracts'.

(II).View Functions

```
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:

The screenshot shows the Truffle UI interface. The 'Function program.sol' tab is active, displaying the following Solidity code:

```
1 pragma solidity ^0.5.0;
2
3 contract Test {
4     Function getResult() public view returns(uint product, uint sum){
5         uint a = 1; // local variable
6         uint b = 2;
7         product = a * b;
8         sum = a + b;
9     }
10 }
```

The UI includes fields for 'ENVIRONMENT' (JavaScript VM), 'ACCOUNT' (0x5B3...eddC4), 'GAS LIMIT' (3000000), and 'VALUE' (0 wei). Below these, there's a 'CONTRACT' section for 'Test - Function_program.sol' with a 'Deploy' button. There are also sections for 'Transactions recorded' and 'Deployed Contracts'.

BLOCKCHAIN

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons and settings: Compiler (version 0.5.17+commit.d19bba13), Language (Solidity), EVM Version (compiler default), Compiler Configuration (Auto compile, Enable optimization set to 200, Hide warnings), and a prominent blue button labeled "Compile Function_program.sol". Below these are sections for CONTRACT (Test (Function_program.sol)) and deployment options: "Publish on Swarm" (with a cloud icon) and "Publish on Ipfs" (with an IPFS icon). At the bottom of the sidebar are links for "Compilation Details", "ABI", and "Bytecode". The main area is a code editor titled "Function_program.sol" with the following Solidity 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;
    }
}
```

(III).Pure Functions

```
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; }

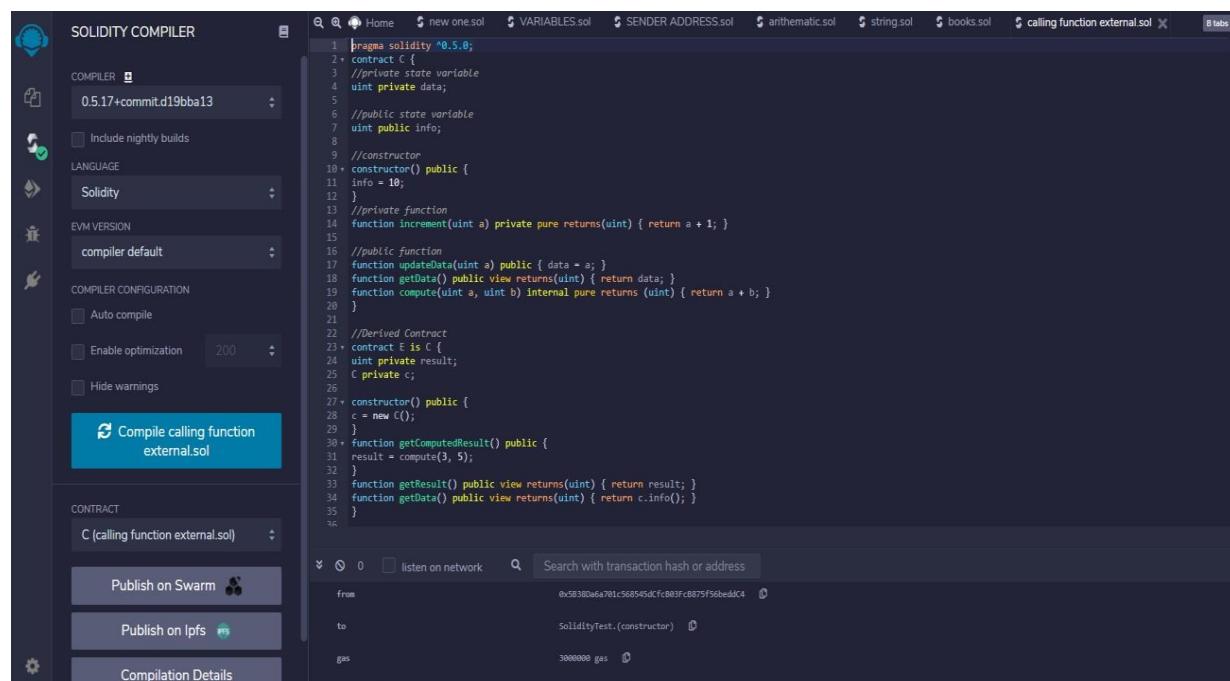
}

//Derived Contract
```

BLOCKCHAIN

```
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(); }  
}
```

Output:



The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with settings for the compiler version (0.5.17+commit.d19bba13), language (Solidity), EVM version (compiler default), and compiler configuration (Auto compile). A prominent blue button at the bottom left says "Compile calling function external.sol". The main area displays the Solidity code for contracts C and E. Below the code, a transaction details section shows a transaction being sent from a specific address to the contract's constructor, with a gas limit of 3000000.

```
1 pragma solidity <=0.5.0;  
2 contract C {  
3     //private state variable  
4     uint private data;  
5  
6     //public state variable  
7     uint public info;  
8  
9     //constructor  
10    constructor() public {  
11        info = 10;  
12    }  
13    //private function  
14    function increment(uint a) private pure returns(uint) { return a + 1; }  
15  
16    //public function  
17    function updateData(uint a) public { data = a; }  
18    function getData() public view returns(uint) { return data; }  
19    function compute(uint a, uint b) internal pure returns (uint) { return a + b; }  
20 }  
21  
22 //Derived Contract  
23 contract E is C {  
24     uint private result;  
25     C private c;  
26  
27     constructor() public {  
28         c = new C();  
29     }  
30     function getComputedResult() public {  
31         result = compute(3, 5);  
32     }  
33     function getResult() public view returns(uint) { return result; }  
34     function getData() public view returns(uint) { return c.info(); }  
35 }
```

from: 0x5838D6a701c568545dCfB83Fc8875f56bedd4C
to: SolidityTest.(constructor)
gas: 3000000 gas

BLOCKCHAIN

The screenshot shows the Truffle UI interface for a Solidity project. The main area displays two contracts: `TEST AT 0X0FC...9A836 (MEMORY)` and `C AT 0XAEO...96B8B (MEMORY)`. The `TEST` contract has a `setBook` function and a `getBookid` function. The `C` contract has an `updateData` function, which is currently being interacted with, showing a value of 25 and a `transact` button. Below these, there are `getData` and `info` buttons. The bottom right corner shows a transaction details panel with fields: `from` (0x58380fa701c568545dCfC903Fc875f5fbbedC4), `to` (SolidityTest.(constructor)), and `gas` (3000000 gas). The top navigation bar includes tabs for Home, new one.sol, VARIABLES.sol, SENDER ADDRESS.sol, arithmetic.sol, string.sol, books.sol, and calling function external.sol.

```
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(); }
}
```

(V).Function Overloading

```
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);
    }
}
```

BLOCKCHAIN

Output:

The screenshot displays the Truffle UI interface, divided into two main sections: SOLIDITY COMPILER and DEPLOY & RUN TRANSACTIONS.

SOLIDITY COMPILER:

- Compiler: 0.5.17+commit.d19bba13
- Include nightly builds:
- Language: Solidity
- EVM Version: compiler default
- Compiler Configuration:
 - Auto compile:
 - Enable optimization: 200
 - Hide warnings:
- Compile overloading.sol

DEPLOY & RUN TRANSACTIONS:

- Contract: Test (overloading.sol)
- Publish options:
 - Publish on Swarm
 - Publish on Ipfs
- Compilation Details
- Deploy & Run Transactions:
 - CALLDATA: Transaction
 - TEST AT 0X7B9_B6ACE (MEMORY):
 - callSumWithT... (highlighted)
 - callSumWithT...
 - getSum:
 - a: 3
 - b: 4
 - callResult: 0: uint256: 7
 - getSum:
 - a: 2
 - b: 3
 - c: 4
 - callResult: 0: uint256: 9
- Transaction details:
 - From: 0x583BDa6a701c568545dCfC803Fc8875f56bedd4
 - To: SolidityTest.(constructor)
 - Gas: 3000000 gas
 - Gas used: 116859 gas

BLOCKCHAIN

(VI).Mathematical Functions

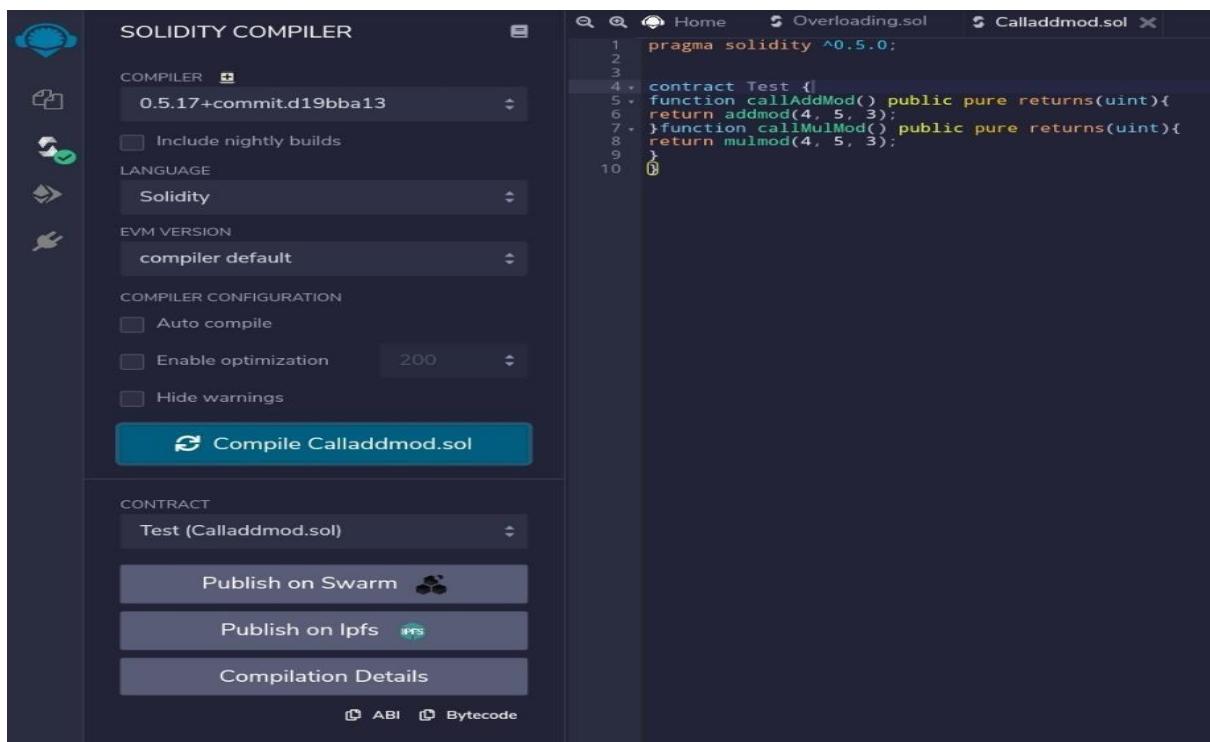
```
pragma solidity ^0.5.0;

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:

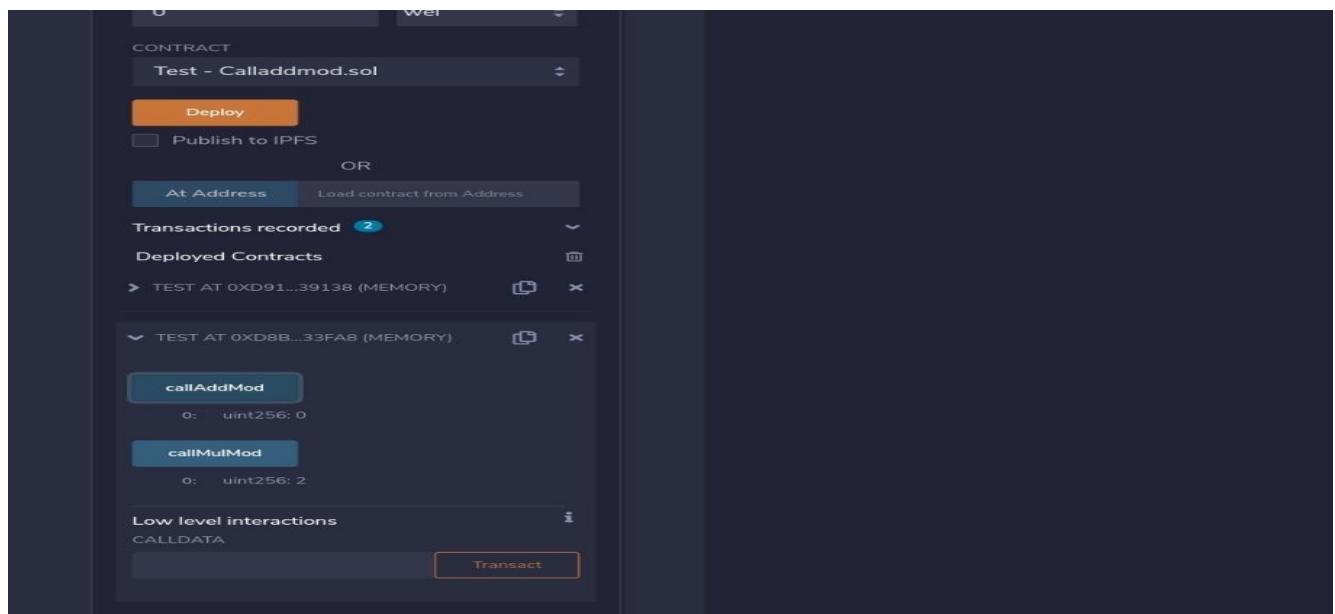


The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons. The main area is titled "SOLIDITY COMPILER". It includes sections for "COMPILER" (set to 0.5.17+commit.d19bba13), "LANGUAGE" (Solidity), "EVM VERSION" (compiler default), and "COMPILER CONFIGURATION" (with options for Auto compile, Enable optimization set to 200, and Hide warnings). A large blue button at the bottom left says "Compile Calladdmod.sol". Below that is a "CONTRACT" section with a dropdown set to "Test (Calladdmod.sol)". Underneath are three buttons: "Publish on Swarm" (with a Swarm icon), "Publish on Ipfs" (with an IPFS icon), and "Compilation Details". At the very bottom are links for ABI and Bytecode.

```
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);
    }
}
```

BLOCKCHAIN



(VII).Cryptographic Functions

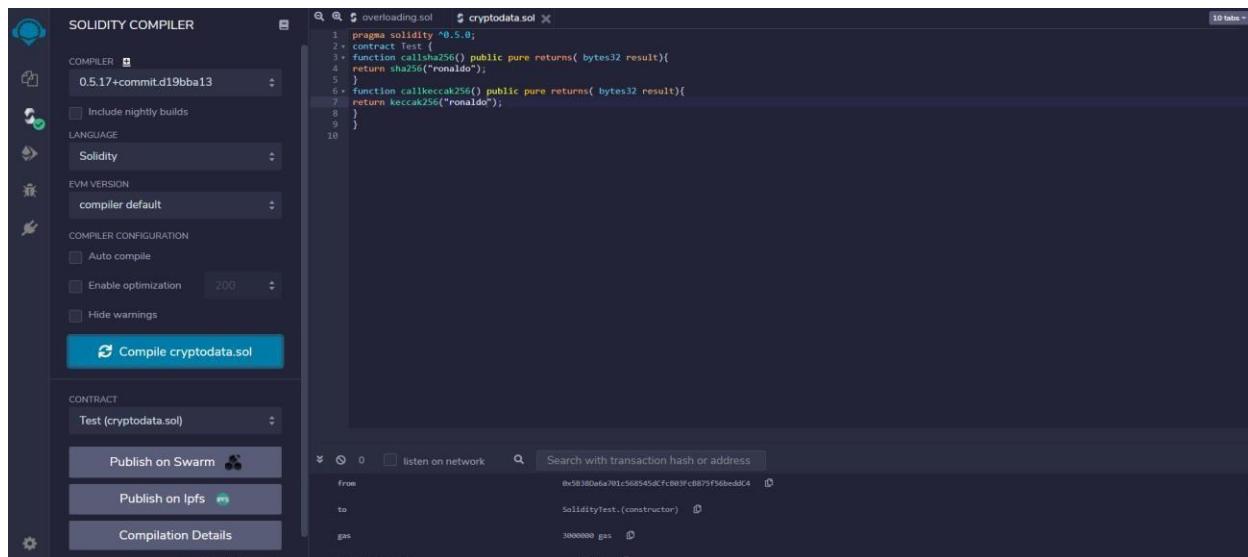
```
pragma solidity ^0.5.0;

contract Test {

    function callsha256() public pure returns( bytes32 result){
        return sha256("ronaldo");
    }

    function callkeccak256() public pure returns( bytes32 result){
        return keccak256("ronaldo");
    }
}
```

Output:



BLOCKCHAIN

The screenshot shows a blockchain development environment with the following components:

- Top Bar:** Shows tabs for "overloading.sol" and "cryptodata.sol". A "10 tabs" indicator is in the top right.
- Left Sidebar:** Includes icons for deployment, running transactions, monitoring, and more. It also lists "Low level interactions" and "CALLDATA".
- Middle Panel:** Displays Solidity code:

```
1 pragma solidity ^0.5.0;
2 + contract Test {
3     function callsha256() public pure returns( bytes32 result){
4         return sha256("ronaldo");
5     }
6     function callkeccak256() public pure returns( bytes32 result){
7         return keccak256("ronaldo");
8     }
9 }
```
- Bottom Panel:** Shows a "TEST AT 0X332...D4B6D (MEMORY)" section with two results:
 - callkeccak256:** Returns bytes32 result: 0x2ce9605c4975b623646a25b4c09ba530f0d4ff92359b673bce0e5538aed8
 - callsha256:** Returns bytes32 result: 0xe24dd2210803b4737a9bd9e31634ac807b63201c3bc32b68fb122c526efff36
- Bottom Right:** Transaction details: "from" (0x58380a6a701c568545dcfc803fc8875f56bedd4), "to" (SolidityTest.(constructor)), "gas" (3000000 gas), and "transaction cost" (116859 gas).

Practical 7

Implement and demonstrate the use of the following in Solidity :

- (I).Contracts
- (II).Inheritance
- (III).Constructors
- (IV).Abstract Class
- (V).Interfaces

(I).Contracts

```
// Solidity program to
// demonstrate how to
// write a smart contract
pragma solidity >= 0.4.16 < 0.7.0;

// Defining a contract
contract Test
{
    // Declaring state variables
    uint public var1;
    uint public var2;
    uint public sum;

    // Defining public function
    // that sets the value of
    // the state variable
    function set(uint x, uint y) public
    {
        var1 = x;
        var2=y;
        sum=var1+var2;
    }
}
```

BLOCKCHAIN

```
// Defining function to  
// print the sum of  
// state variables  
function get()  
) public view returns (uint) {  
return sum;  
}  
}
```

Output:

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various settings like compiler version (0.5.17+commit.d19ba13), language (Solidity), EVM version (compiler default), and compiler configuration options. The main area displays the Solidity code. Below the code, there are sections for publishing the contract to Swarm or IPFS, and a 'Compilation Details' section.

```
// Solidity program to  
// demonstrate how to  
// write a smart contract  
pragma solidity >= 0.4.16 < 0.7.0;  
  
// Defining a contract  
contract Test  
{  
  
    // Declaring state variables  
    uint public var1;  
    uint public var2;  
    uint public sum;  
  
    // Defining public function  
    // that sets the value of  
    // the state variable  
    function set(uint x, uint y) public  
    {  
        var1 = x;  
        var2=y;  
        sum=var1+var2;  
    }  
  
    // Defining function to  
    // print the sum of  
    // state variables  
    function get()  
) public view returns (uint) {  
return sum;  
}  
}
```

COMPILER
0.5.17+commit.d19ba13
Include nightly builds

LANGUAGE
Solidity

EVM VERSION
compiler default

COMPILER CONFIGURATION
Auto compile
Enable optimization 200
Hide warnings

CONTRACT
Test (smartcontract.sol)

Publish on Swarm
Publish on Ipfs

Compilation Details

from 0x5b30a6a701c568545dcfc883fc8875f56bedd04
to SolidityTest.(constructor)
gas 3000000 gas

BLOCKCHAIN

```
// Solidity program to
// demonstrate how to
// write a smart contract
pragma solidity >=0.4.16 < 0.7.0;

// Defining a contract
contract Test {
    // Declaring state variables
    uint public var1;
    uint public var2;
    uint public sum;

    // Defining public function
    // that sets the value of
    // the state variable
    function set(uint x, uint y) public {
        var1 = x;
        var2=y;
        sum=var1+var2;
    }

    // Defining function to
    // print the sum of
    // state variables
    function get() public view returns (uint) {
        return sum;
    }
}
```

Low level interactions

TEST AT 0x5e1...4eff5 (MEMORY)

set

x: 20

y: 15

transaction

get

sum

var1

var2

Low level interactions

0x5e1...4eff5 from: 0x5b3...edd4 to: Test.set(uint256,uint256) 0x5e1...4eff5 value: 0 wei data: 0x1ab...0000f logs: 0 hash: 0x85f...143c6

status: true Transaction mined and execution succeed

transaction hash: 0x85f...143c6

Search with transaction hash or address

(II).Inheritance

```
// Solidity program to demonstrate Single Inheritance

pragma solidity >=0.4.22 <0.6.0;

// Defining contract
contract parent{

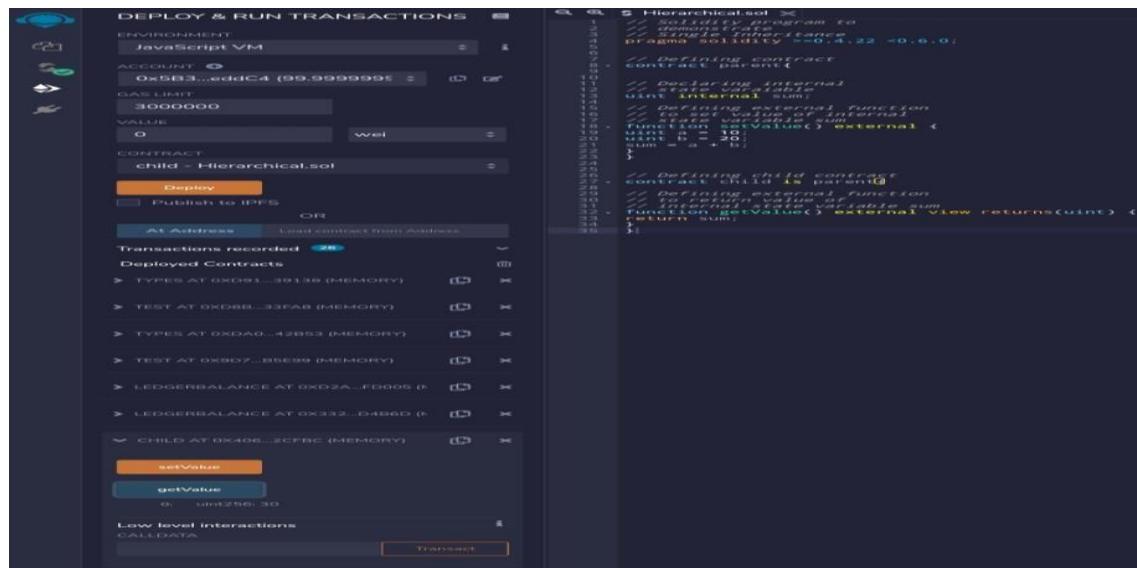
    // Declaring internal state varaiable
    uint internal sum;

    // Defining external function to set value of internal state variable sum
    function setValue() external {
        uint a = 10;
        uint b = 20;
        sum = a + b;
    }
}
```

BLOCKCHAIN

```
// Defining child contract  
  
contract child is parent{  
  
// Defining external function to return value of internal state variable sum  
  
function getValue() external view returns(uint) {  
  
return sum;  
  
}  
  
}  
  
// Defining calling contract  
  
contract caller {  
  
// Creating child contract object  
  
child cc = new child();  
  
// Defining function to call setValue and getValue functions  
  
function testInheritance() public {  
  
cc.setValue();  
  
}  
  
function result() public view returns(uint ){  
  
return cc.getValue();  
  
}  
  
}
```

Output:



The screenshot shows the Truffle UI interface with two main panes. The left pane, titled 'DEPLOY & RUN TRANSACTIONS', displays the deployment configuration for a Solidity contract named 'child - Hierarchical.sol'. It includes fields for ACCOUNT (0x5B3...eaddC4), GAS LIMIT (3000000), and VALUE (0 wei). Below these, under 'CONTRACT', is the selected contract 'child - Hierarchical.sol'. There are buttons for 'Deploy' and 'Publish to IPFS'. Under 'Transactions recorded', there are several entries: 'TYPES AT 0XD91...39138 (MEMORY)', 'TEST AT 0XD88L...33FA8 (MEMORY)', 'TEST AT 0XDA0...42853 (MEMORY)', 'TEST AT 0XB07...B5E99 (MEMORY)', 'LEDGERBALANCE AT 0XD2A...FD0005 (0)', and 'LEDGERBALANCE AT 0X332...D496D (0)'. A section for 'CHILD AT 0X406...2CFBC (MEMORY)' shows a 'setValue' button and a 'getValue' button with a value of 0. Under 'Low level interactions', there is a 'CALLDATA' field and a 'Transact' button. The right pane, titled 'Hierarchical.sol', displays the Solidity code with line numbers. The code defines a parent contract with an internal state variable 'sum' and an external function 'getValue()'. It also defines a child contract that inherits from the parent and overrides the 'getValue()' function to return the sum of two variables 'a' and 'b'.

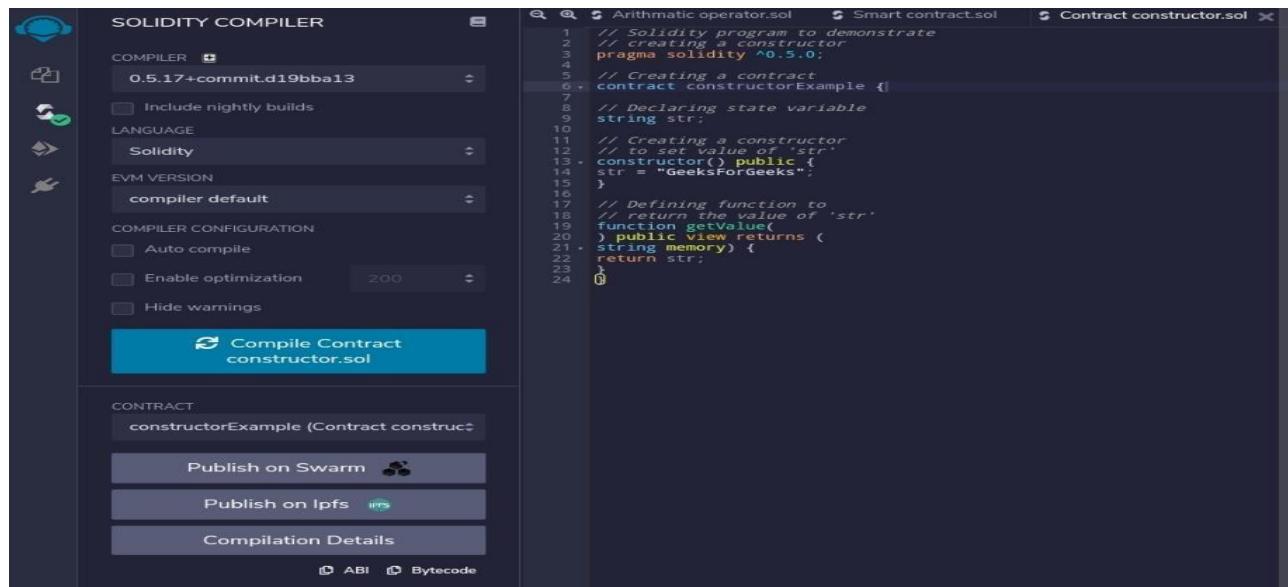
```
1 // Solidity program to  
2 // demonstrate inheritance  
3 // Contract solidity v0.4.22 <0.6.0;  
4  
5 // DEFINING CONTRACT  
6 CONTRACT parent{  
7  
8 // Declaring internal state variable sum  
9 MINT INTERNAL;  
10 // Defining external function  
11 // returns state variable sum  
12 function getValue() external {  
13     uint a = 10;  
14     uint b = 20;  
15     sum = a + b;  
16 }  
17  
18 // DEFINING CHILD contract  
19 CONTRACT child IS parent{  
20  
21 // Defining external function  
22 // returns internal state variable sum  
23 function overrideValue() external View returns(uint) {  
24     return sum;  
25 }  
26 }
```

BLOCKCHAIN

(III).Constructors

```
// Solidity program to demonstrate  
// creating a constructor  
pragma solidity ^0.5.0;  
  
// Creating a contract  
contract constructorExample {  
  
    // Declaring state variable  
    string str;  
  
    // Creating a constructor  
    // to set value of 'str'  
    constructor() public {  
        str = "GeeksForGeeks";  
    }  
  
    // Defining function to  
    // return the value of 'str'  
    function getValue()  
        public view returns (  
            string memory) {  
        return str;  
    }  
}
```

Output:



The screenshot shows the Solidity Compiler interface with the following details:

- SOLIDITY COMPILER** tab is selected.
- COMPILER**: 0.5.17+commit.d19bba13
- LANGUAGE**: Solidity
- EVM VERSION**: compiler default
- COMPILER CONFIGURATION**: Auto compile, Enable optimization (200), Hide warnings
- Compile Contract** button (highlighted in blue)
- CONTRACT**: constructorExample (Contract constructor)
- Publish on Swarm** button
- Publish on Ipfs** button (green)
- Compilation Details** button
- ABI** and **Bytecode** links at the bottom

The code editor on the right displays the Solidity code for 'Contract constructor.sol'.

```
// Solidity program to demonstrate  
// creating a constructor  
pragma solidity ^0.5.0;  
  
// Creating a contract  
contract constructorExample {  
  
    // Declaring state variable  
    string str;  
  
    // Creating a constructor  
    // to set value of 'str'  
    constructor() public {  
        str = "GeeksForGeeks";  
    }  
  
    // Defining function to  
    // return the value of 'str'  
    function getValue()  
        public view returns (  
            string memory) {  
        return str;  
    }  
}
```

BLOCKCHAIN

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.9999999)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: constructorExample - Contract constru

Deploy

Publish to IPFS

OR

At Address: Load contract from Address

Transactions recorded: 6

Deployed Contracts:

- SOLIDITYTEST AT 0xD91...39138 (MEMORY)
- SOLIDITYTEST AT 0xD8B...33FA8 (MEMORY)
- SOLIDITYTEST AT 0xF8E...9FBEB (MEMORY)
- TEST AT 0xD7A...F771B (MEMORY)
- CONSTRUCTOREXAMPLE AT 0xDA0...4 (MEMORY)

getValue

0: string: GeeksForGeeks

Low level interactions:

CALLDATA:

Transact

Arithmatic operator.sol Smart contract.sol Contract constructor.sol

```
1 // Solidity program to demonstrate
2 // creating a constructor
3 pragma solidity ^0.5.0;
4
5 // Creating a contract
6 contract constructorExample {
7
8 // Declaring state variable
9 string str;
10
11 // Creating a constructor
12 // to set value of 'str'
13 constructor() public {
14 str = "GeeksForGeeks";
15 }
16
17 // Defining function to
18 // return the value of 'str'
19 function getValue()
20 public view returns (
21 string memory) {
22 return str;
23 }
24 }
```

BLOCKCHAIN

Practical 8

Implement and demonstrate the use of the following in Solidity :

(I).Libraries

(II).Assembly

(III).Events

(IV).Error Handling

(IV).Error Handling

```
// Solidity program to demonstrate require statement
pragma solidity ^0.5.0;

// Creating a contract
contract requireStatement {
    // Defining function to check input
    function checkInput(
        uint _input) public view returns(
        string memory){
        require(_input >= 0, "invalid uint8");
        require(_input <= 255, "invalid uint8");
        return "Input is Uint8";
    }

    // Defining function to use require statement
    function Odd(uint _input) public view returns(bool){
        require(_input % 2 != 0);
        return true;
    }
}
```

Output:

BLOCKCHAIN

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

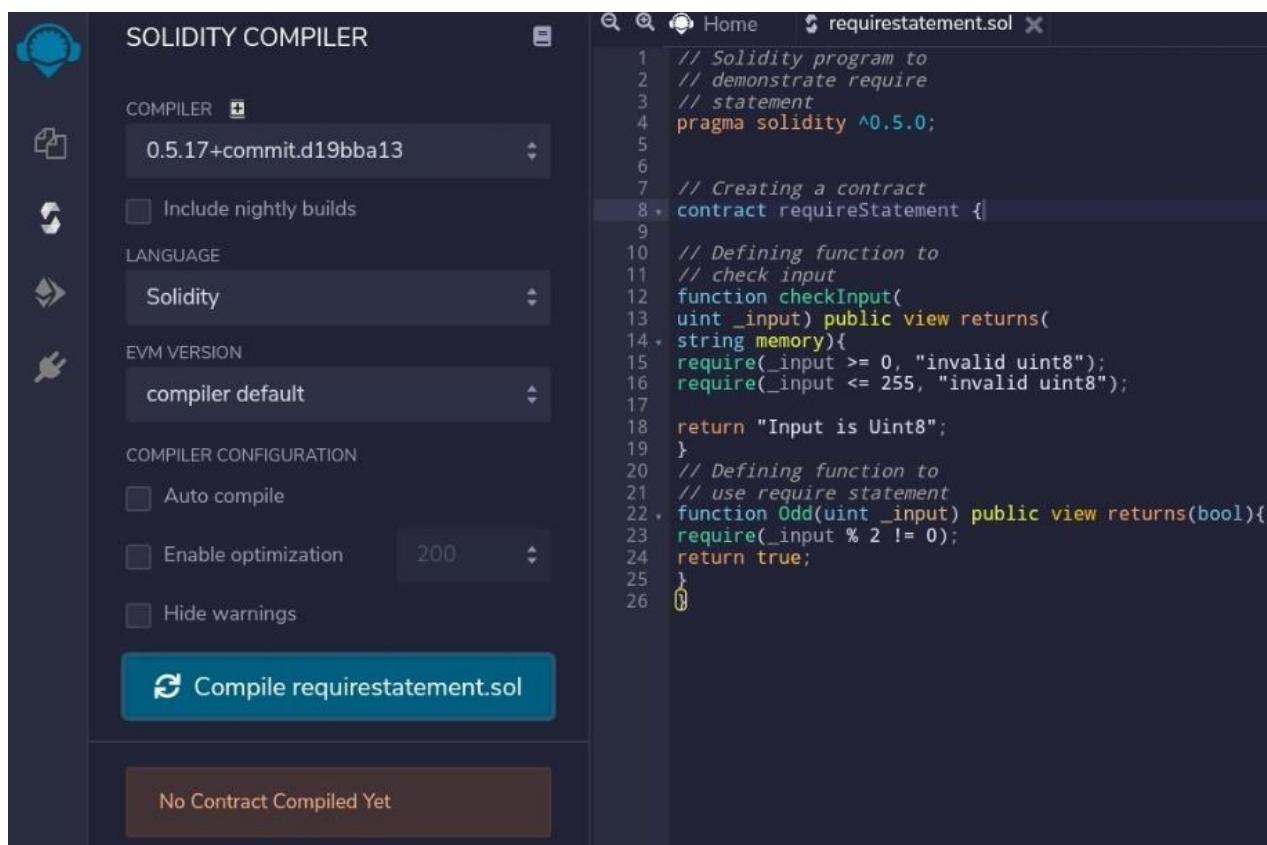
LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILE CONFIGURATION: Auto compile, Enable optimization (200), Hide warnings

Compile requirestatement.sol

No Contract Compiled Yet



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.9999999)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: requireStatement - requirestatement.sol

Deploy

Publish to IPFS

OR

At Address: Load contract from Address

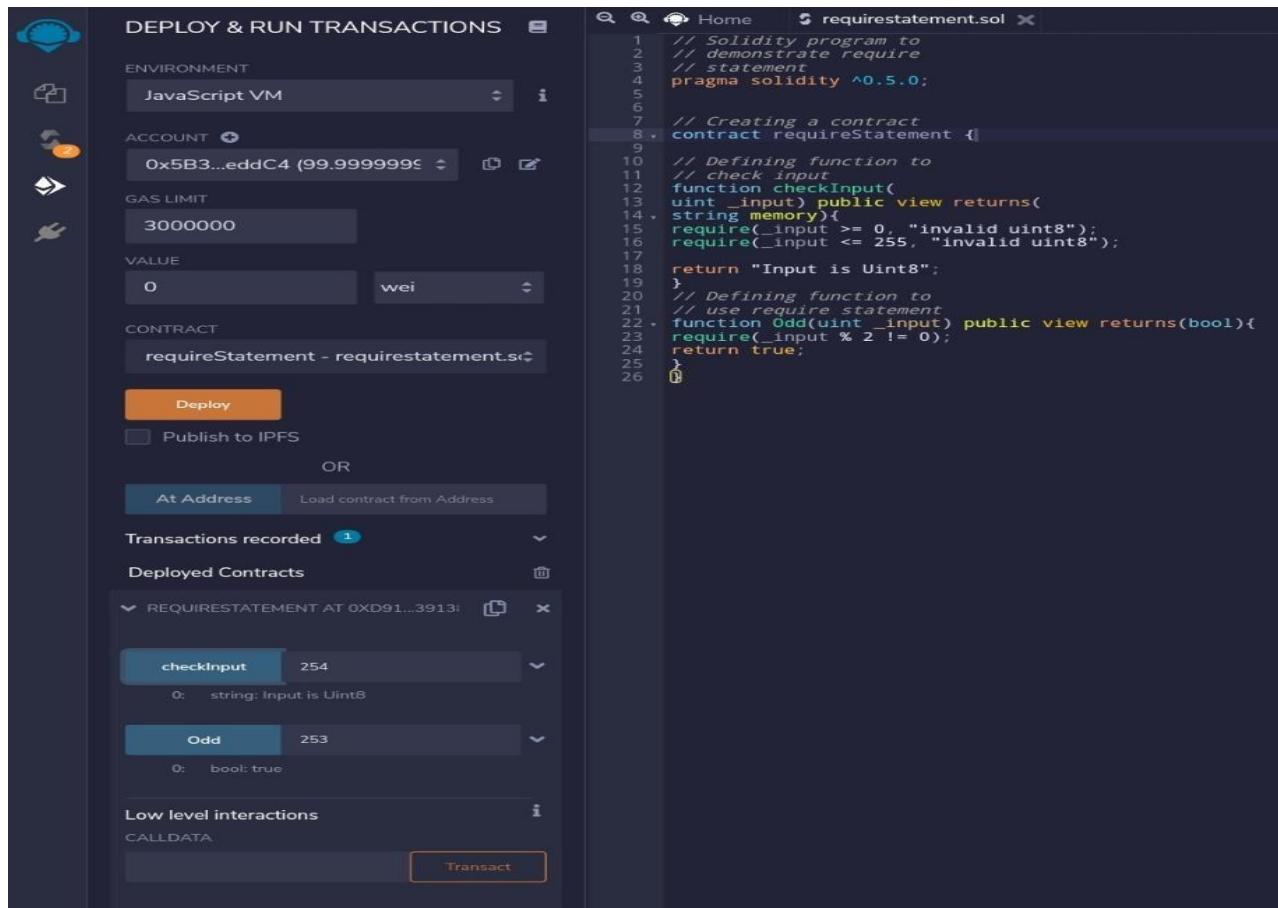
Transactions recorded: 1

Deployed Contracts:

- REQUIRESTATEMENT AT 0xD91...3913: checkInput (254), Odd (253)

Low level interactions: CALLDATA

Transact



BLOCKCHAIN

```
// Solidity program to demonstrate assert statement
```

```
pragma solidity ^0.5.0;
```

```
// Creating a contract
```

```
contract assertStatement {
```

```
// Defining a state variable
```

```
bool result;
```

```
// Defining a function to check condition
```

```
function checkOverflow(
```

```
uint _num1, uint _num2) public {
```

```
uint8 sum = _num1 + _num2;
```

```
assert(sum<=255);
```

```
result = true;
```

```
}
```

```
// Defining a function to print result of assert statement
```

```
function getResult() public view returns(string memory){
```

```
if(result == true){
```

```
return "No Overflow";
```

```
}
```

```
else{
```

```
return "Overflow exist";
```

```
}
```

```
}
```

```
}
```

Output:

BLOCKCHAIN

The screenshot shows the Solidity Compiler interface. On the left, there are several icons: a brain, a gear, a checkmark, a magnifying glass, a plus sign, a gear, a lightning bolt, and a hand. The main area has tabs for 'Solidity Compiler' (selected), 'Compiler Help', and 'Solidity Language'. Under 'SolidITY COMPILER', the version is listed as '0.5.17+commit.d19bba13'. Below it are sections for 'Include nightly builds', 'LANGUAGE' set to 'Solidity', and 'EVM VERSION' set to 'compiler default'. Under 'COMPILER CONFIGURATION', there are three checkboxes: 'Auto compile', 'Enable optimization' (set to 200), and 'Hide warnings'. A large blue button at the bottom left says 'Compile assertstatement.sol'. The right side shows the code for 'assertstatement.sol':

```
// Solidity program to demonstrate assert statements
pragma solidity ^0.5.0;

// Creating a contract
contract assertstatement {
    // Defining a state variable
    bool result;
}

// Defining a function to check condition
function checkoverflow(
    uint8 _num1, uint8 _num2) public {
    uint8 sum = _num1 + _num2;
    assert(sum<=255);
    result = true;
}

// Defining a function to print result of assert statement
function getResult() public view returns(string memory){
    if(result==true)
        return "No Overflow";
    else
        return "Overflow exist";
}
}
```

BLOCKCHAIN

```
// Solidity program to demonstrate assert statement
pragma solidity ^0.5.0;

// Creating a contract
contract assertStatement {

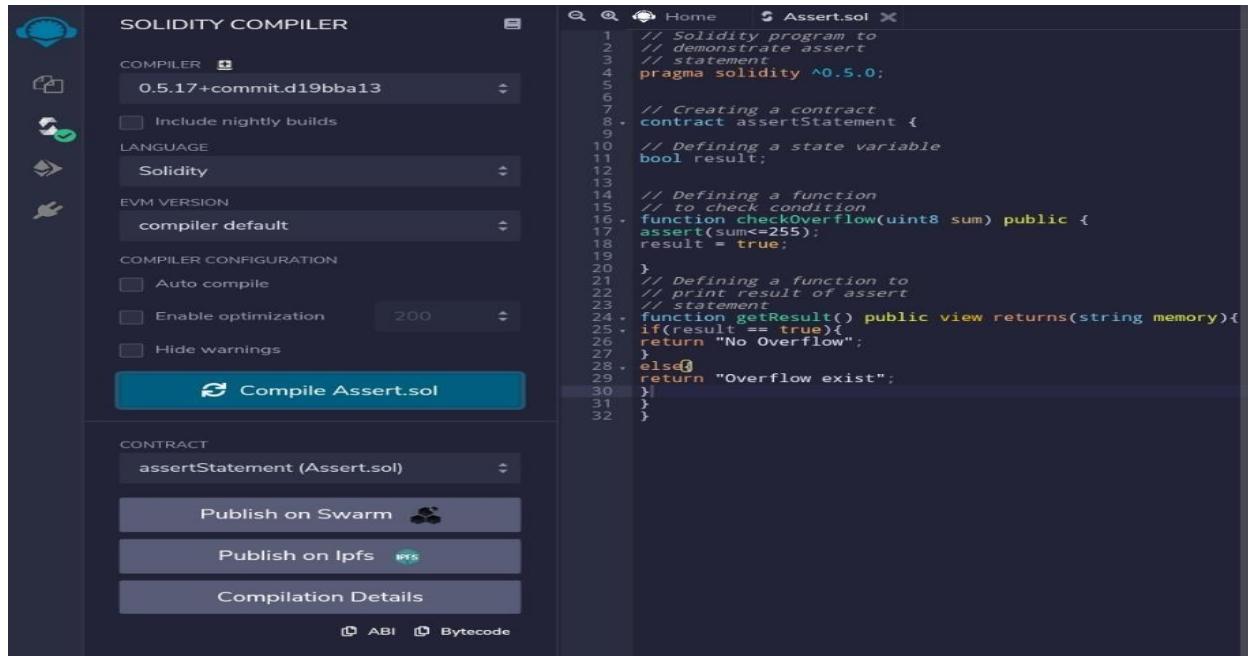
    // Defining a state variable
    bool result;

    // Defining a function
    // to check condition
    function checkOverflow(uint8 sum) public {
        assert(sum<=255);
        result = true;
    }

    // Defining a function to print result of assert statement
    function getResult() public view returns(string memory){
        if(result == true){
            return "No Overflow";
        }
        else{
            return "Overflow exist";
        }
    }
}
```

Output:

BLOCKCHAIN



The Solidity Compiler interface shows the compilation of a contract named `assertStatement (Assert.sol)`. The code is as follows:

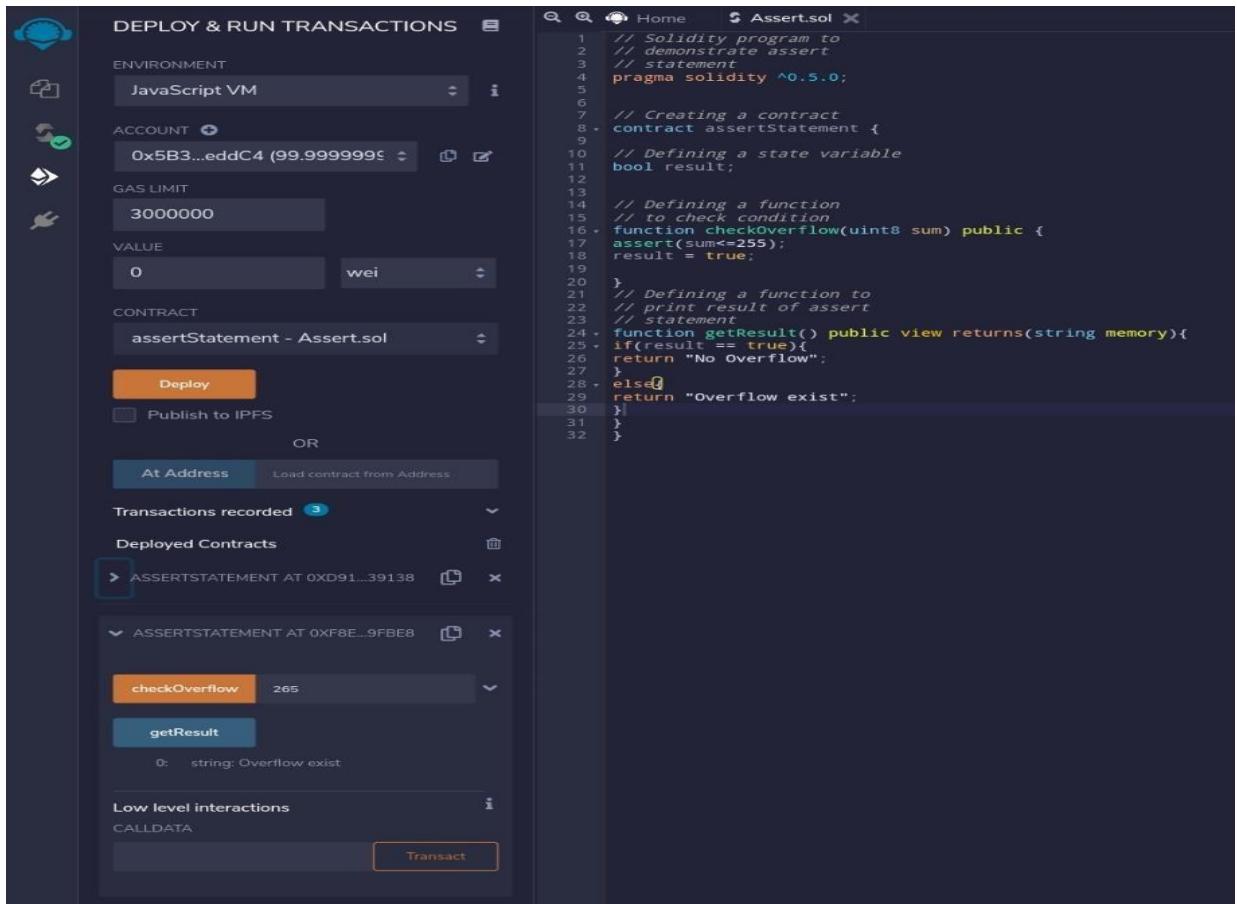
```
// Solidity program to
// demonstrate assert
// statement
pragma solidity ^0.5.0;

// Creating a contract
contract assertStatement {
    // Defining a state variable
    bool result;

    // Defining a function
    // to check condition
    function checkOverflow(uint8 sum) public {
        assert(sum<=255);
        result = true;
    }

    // Defining a function to
    // print result of assert
    // statement
    function getResult() public view returns(string memory){
        if(result == true){
            return "No Overflow";
        }
        else{
            return "Overflow exist";
        }
    }
}
```

Compiler settings include EVM version `compiler default`, optimization level `200`, and ABI and Bytecode output options.



The Deploy & Run Transactions interface shows the deployment of the `assertStatement - Assert.sol` contract to a JavaScript VM environment. The deployed contract address is `0xD91...39138`.

The interface also displays the `checkOverflow` and `getResult` functions. The `getResult` function is called with a value of 265, returning the string `Overflow exist`.

Low-level interactions and call data are also visible.

BLOCKCHAIN

```
// Solidity program to demonstrate revert statement
pragma solidity ^0.5.0;

// Creating a contract
contract revertStatement {

    // Defining a function to check condition
    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:

BLOCKCHAIN

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILE CONFIGURATION: Auto compile, Enable optimization (200), Hide warnings

Compile revertstatement.sol

CONTRACT: revertStatement (revertstatement.sol)

Publish on Swarm, Publish on lpf5, Compilation Details

Search with transaction hash or address

[vm] from: 0x583...eddC4 to: Test.set(uint256,uint256) 0x5e1...4Eeff5 value: 0 wei data: 0x1ab...0000f logs: 0 hash: 0x85f...143c6

status: true Transaction mined and execution succeed

transaction hash: 0x85fca5de9819161eb66c32b59aab8e9c4276740d75c05a17394c2854b3143c6

Debug

```
// Solidity program to demonstrate revert statement
pragma solidity ^0.5.0;

// Creating a contract
contract revertStatement {

    // Defining a function to check condition
    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);
        }
    }
}
```

DEPLOY & RUN TRANSACTIONS

var1, var2

Low level interactions

CALLDATA: Transact

REVERTSTATEMENT AT 0x1C9...2B4BC

checkOverflow

_num1: 52, _num2: 35

call

0: string: No Overflow

1: uint256: 87

Low level interactions

CALLDATA: Transact

Search with transaction hash or address

[vm] from: 0x583...eddC4 to: Test.set(uint256,uint256) 0x5e1...4Eeff5 value: 0 wei data: 0x1ab...0000f logs: 0 hash: 0x85f...143c6

status: true Transaction mined and execution succeed

transaction hash: 0x85fca5de9819161eb66c32b59aab8e9c4276740d75c05a17394c2854b3143c6

Debug

```
// Solidity program to demonstrate revert statement
pragma solidity ^0.5.0;

// Creating a contract
contract revertStatement {

    // Defining a function to check condition
    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);
        }
    }
}
```