

MSC.IT PART II

BLOCKCHAIN PRACTICAL JOURNAL

Index

Sr.No	Topic	Date	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		

Index

Sr.No	Topic	Date	Remark
6	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		
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)

    @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()

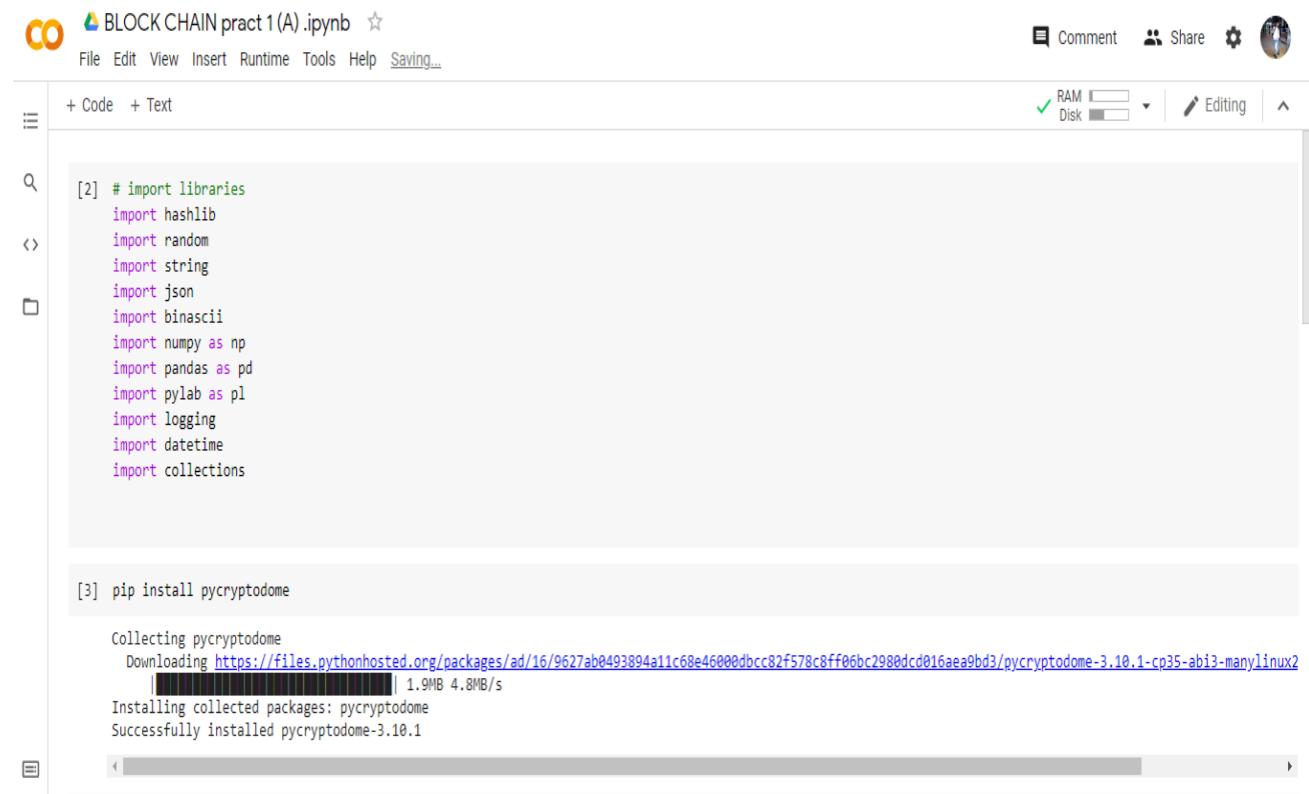
t = Transaction(Dinesh,Ramesh.identity,5.0)

signature = t.sign_transaction()

print (signature)

```

Output:



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** CO BLOCK CHAIN pract 1(A).ipynb
- Toolbar:** File Edit View Insert Runtime Tools Help Saving...
- Code Cell 2:**

```
[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
```
- Code Cell 3:**

```
[3] pip install pycryptodome
Collecting pycryptodome
  Downloading https://files.pythonhosted.org/packages/ad/16/9627ab0493894a11c68e46000dbcc82f578c8ff06bc2980dc016aea9bd3/pycryptodome-3.10.1-cp35-abi3-manylinux2
    |████████| 1.9MB 4.8MB/s
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.10.1
```
- Runtime Status:** RAM 75% Disk 10% Editing

BLOCK CHAIN pract1(A).ipynb

```
[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()
```

BLOCK CHAIN pract1(A).ipynb

```
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)
```

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(

 Dinesh,

 Seema.identity,

 6.0

)

```
t2.sign_transaction()  
transactions.append(t2)
```

t3 = Transaction(

 Ramesh,

 Vijay.identity,

 2.0

)

```
t3.sign_transaction()  
transactions.append(t3)
```

t4 = Transaction(

 Seema,

 Ramesh.identity,

 4.0

)

```
t4.sign_transaction()  
transactions.append(t4)
```

t5 = Transaction(

 Vijay,

 Seema.identity,

```
    7.0
)
t5.sign_transaction()
transactions.append(t5)

t6 = Transaction(
    Ramesh,
    Seema.identity,
    3.0
)
t6.sign_transaction()
transactions.append(t6)

t7 = Transaction(
    Seema,
    Dinesh.identity,
    8.0
)
t7.sign_transaction()
transactions.append(t7)

t8 = Transaction(
    Seema,
    Ramesh.identity,
    1.0
)
t8.sign_transaction()
transactions.append(t8)

t9 = Transaction(
    Vijay,
    Dinesh.identity,
    5.0
```

```
)  
t9.sign_transaction()  
transactions.append(t9)  
t10 = Transaction(  
    Vijay,  
    Ramesh.identity,  
    3.0  
)  
t10.sign_transaction()  
transactions.append(t10)
```

for transaction in transactions:

```
    display_transaction (transaction)  
    print ('-----')
```

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)

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 ('=====')  

    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)
```

```

[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 = ""

```



+ 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  
  
# 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:

```

[ ] # 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
[ ] 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)

251f28f6f523733739979611b404c755210b5b6a70f28d5eb3e3049f7dceea873e472f0df41a55152c71bb6f5

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

```

```

[ ] 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)

```

```

+ 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: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbad86
-----
    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: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbad86384
-----
    recipient: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac49
-----
    value: 4.0
-----
    time: 2021-04-08 06:11:40.570658
-----
    sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
-----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbad86
-----
    value: 7.0
-----
    time: 2021-04-08 06:11:40.572173
-----
    sender: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac492d1
-----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbad86
-----
    value: 3.0
-----
    2021-04-08 06:11:40.573006
[ ] 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)

```

```

+ 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

```

+ Code + Text | 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:

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

LANGUAGE: Solidity

EVM VERSION: compiler default

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

Contract: SolidityTest (new one.sol)

Publish on Swarm, **Publish on IPFS**, **Compilation Details**

DEPLOY & RUN TRANSACTIONS

Value: 3000000 wei

Contract: SolidityTest - new one.sol

Deploy, **Publish to IPFS**

OR

At Address: Load contract from Address

Transactions recorded: 3

Deployed Contracts: SOLIDITYTEST AT 0xD91_39138 [MEM]

getResult: 0: uint256: 10

Low level interactions: CALLDATA, Transact

The screenshot shows the Truffle UI interface. On the left, the 'SOLIDITY COMPILER' section is visible with settings for compiler version (0.5.17+commit.d19bba13), language (Solidity), EVM version (compiler default), and compilation configuration (Auto compile, Enable optimization set to 200, Hide warnings). Below this is the 'CONTRACT' section for 'SolidityTest (new one.sol)'. It includes options to publish to Swarm or IPFS, and a 'Compilation Details' button. On the right, the 'DEPLOY & RUN TRANSACTIONS' section is shown. It has fields for Value (3000000 wei), Contract (SolidityTest - new one.sol), and a 'Deploy' button. There's also an option to Publish to IPFS. Below these are tabs for 'At Address' and 'Transactions recorded' (3). The 'Deployed Contracts' section lists 'SOLIDITYTEST AT 0xD91_39138 [MEM]'. Underneath, a transaction record for 'getResult' is shown with value 0 and type uint256: 10. The bottom part of the interface shows 'Low level interactions' with 'CALLDATA' and 'Transact' buttons.

DEPLOY & RUN TRANSACTIONS

Value: 3000000 wei

Contract: SolidityTest - new one.sol

Deploy, **Publish to IPFS**

OR

At Address: Load contract from Address

Transactions recorded: 3

Deployed Contracts: SOLIDITYTEST AT 0xD91_39138 [MEM]

getResult: 0: uint256: 10

Low level interactions: CALLDATA, Transact

This screenshot is identical to the one above, showing the Truffle UI interface. It displays the Solidity Compiler settings, the deployed contract 'SolidityTest' at address 0xD91_39138, and a transaction record for 'getResult' with value 0. The layout and components are the same as in the first screenshot.

```

// 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;
    }
}

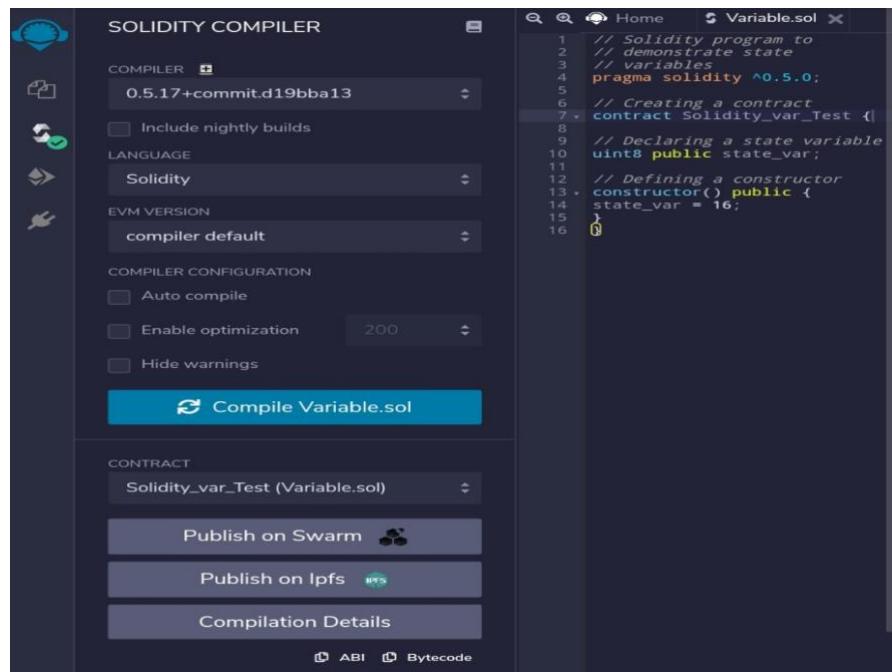
```

Output:

The screenshot shows the Truffle UI interface. On the left, the 'SOLIDITY COMPILER' sidebar is open, displaying compiler settings like version 0.5.17+commit.d19bba13, language Solidity, and EVM version compiler default. It also includes compiler configuration options such as Auto compile, Enable optimization (set to 200), and Hide warnings. A prominent blue button at the bottom of this sidebar says 'Compile new one.sol'. Below the sidebar, the 'CONTRACT' section shows 'Solidity.var_Test (new one.sol)' selected. Underneath are buttons for 'Publish on Swarm' and 'Publish on Ipfs'. At the very bottom is a 'Compilation Details' button.

The main area of the screen displays the Solidity code for 'Solidity.var_Test'. The code is identical to the one shown in the previous code block. To the right of the code, there is a transaction history table with columns for From, to, and gas. One transaction is listed:

From	to	gas
0xc8380a6a701c568545dCfc883FcB875f56beddK4	SolidityTest.(constructor)	3800000 gas



The Solidity Compiler interface shows the following configuration:

- COMPILER:** 0.5.17+commit.d19bba13
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:**
 - Auto compile
 - Enable optimization (set to 200)
 - Hide warnings
- CONTRACT:** Solidity_var_Test (Variable.sol)
- Buttons:** Publish on Swarm, Publish on Ipfs, Compilation Details
- ABI and Bytecode:** ABI and Bytecode links at the bottom.

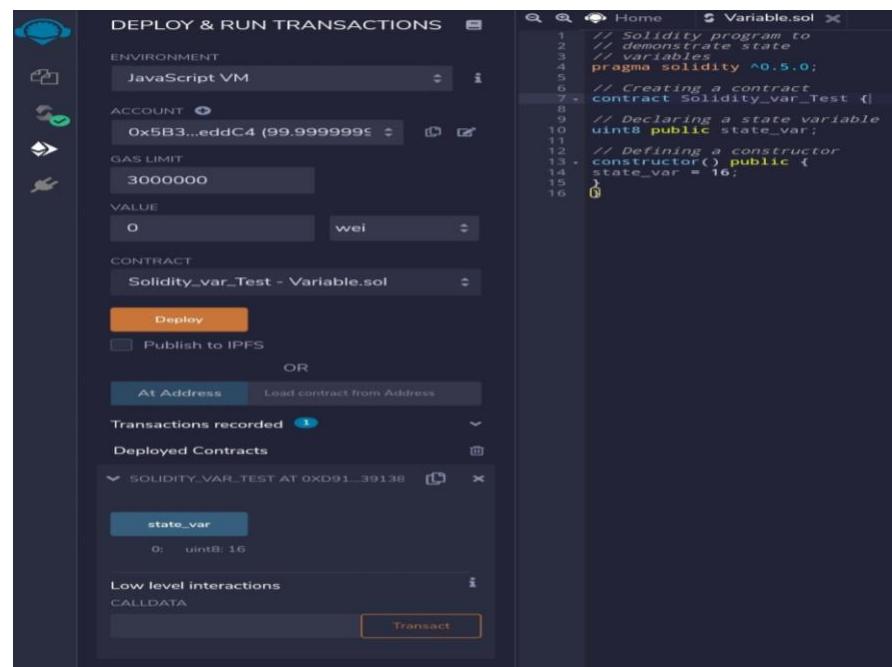
The code editor on the right contains the following Solidity code:

```
// 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 Deploy & Run Transactions interface shows the following configuration:

- ENVIRONMENT:** JavaScript VM
- ACCOUNT:** 0x5B3...eddC4 (99.9999995)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** Solidity_var_Test - Variable.sol
- Buttons:** Deploy, Publish to IPFS
- OR:** At Address, Load contract from Address
- Transactions recorded:** 1
- Deployed Contracts:** SOLIDITY_VAR_TEST AT 0xD91...39138
- Low level interactions:** state_var (0: uint8: 16)
- Calldata:** Transact button

The code editor on the right contains the same Solidity code as the previous screenshot.

```
// 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;
    }
}
```

```
// 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;
```

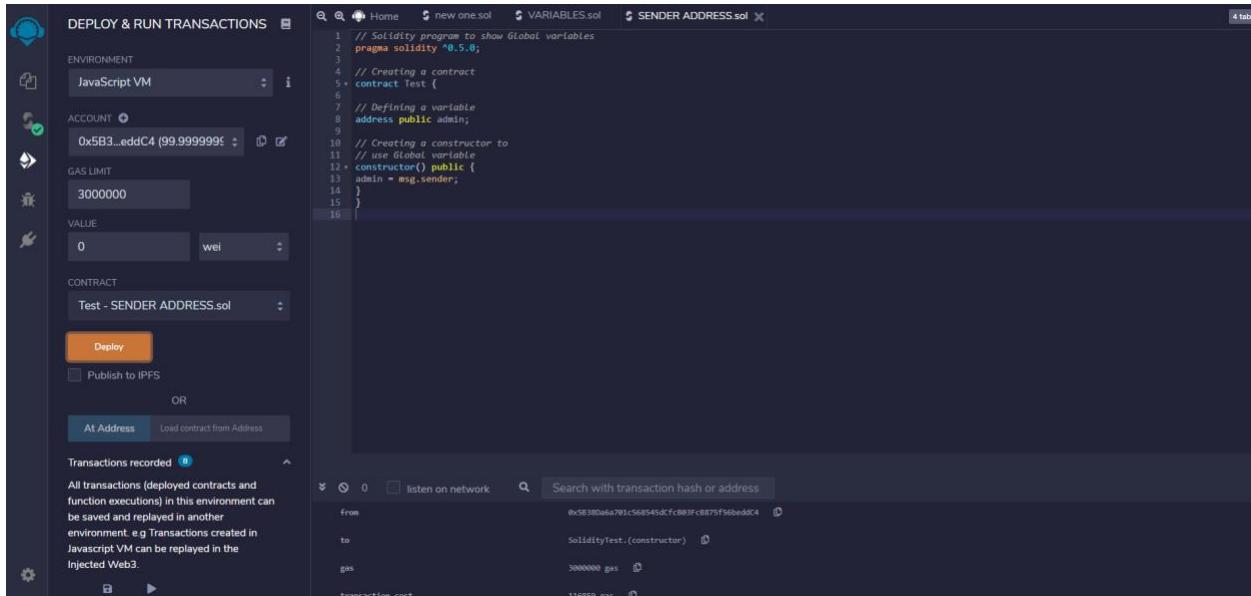
```
}
```

```
}
```

Output:

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

- 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:
- Contract**: Test (SENDER ADDRESS.sol)
 - Publish on Swarm
 - Publish on Ipfs
 - Compilation Details
- Search Bar**: Search with transaction hash or address
- Transaction Fields**:
 - From: 0x513BDa6a701c608545dCfcB83Fc8E75f96b6dC4
 - To: SolidityTest.(constructor)
 - Gas: 3000000 gas



(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;

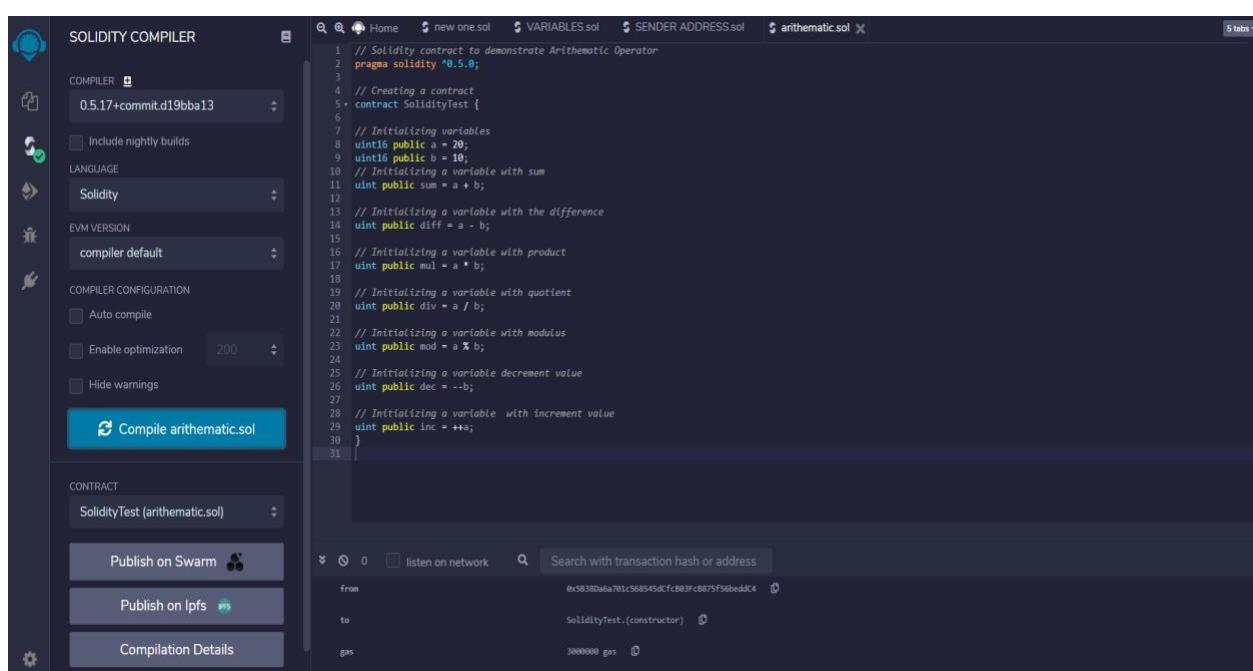
// Initializing a variable with modulus
uint public mod = a % b;

// 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 with the following details:

- SOLIDITY COMPILER** tab is selected.
- COMPILER**: 0.5.17+commit.d19bba13
- LANGUAGE**: Solidity
- EVM VERSION**: compiler default
- COMPILE** button: Compile arithmetic.sol
- CONTRACT**: SolidityTest (arithmetic.sol)
- PUBLISH** buttons: Publish on Swarm, Publish on IPFS
- Compilation Details** section:
 - from: 0x5B3Ba6a701c568545dCfc003fc8875f56bedd4
 - to: SolidityTest.(constructor)
 - gas: 3000000

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.9999995)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: SolidityTest - arithmetic.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 0

All transactions (deployed contracts and function executions) in this environment can be saved and replayed in another environment. e.g. Transactions created in Javascript VM can be replayed in the Injected Web3.

0x5B30a6a701c568545dFcB03FcB875f56beddC4

0x5B30a6a701c568545dFcB03FcB875f56beddC4

3000000 gas

```

1 // Solidity contract to demonstrate Arithmetic Operator
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5 contract SolidityTest {
6
7 // Initializing variables
8 uint16 public a = 20;
9 uint16 public b = 10;
10 // Initializing a variable with sum
11 uint public sum = a + b;
12
13 // Initializing a variable with the difference
14 uint public diff = a - b;
15
16 // Initializing a variable with product
17 uint public mul = a * b;
18
19 // Initializing a variable with quotient
20 uint public div = a / b;
21
22 // Initializing a variable with modulus
23 uint public mod = a % b;
24
25 // Initializing a variable decrement value
26 uint public dec = --b;
27
28 // Initializing a variable with increment value
29 uint public inc = ++b;
30
31

```

DEPLOY & RUN TRANSACTIONS

SOLIDITYTEST AT 0xD2A...FD005 (MEMORY)

a

b

dec

diff

div

inc

mod

mul

sum

Low level interactions

CALldata

Transaction receipt

Transact

CALLDATA

transaction receipt

0x5B30a6a701c568545dFcB03FcB875f56beddC4

0x5B30a6a701c568545dFcB03FcB875f56beddC4

3000000 gas

116850 gas

```

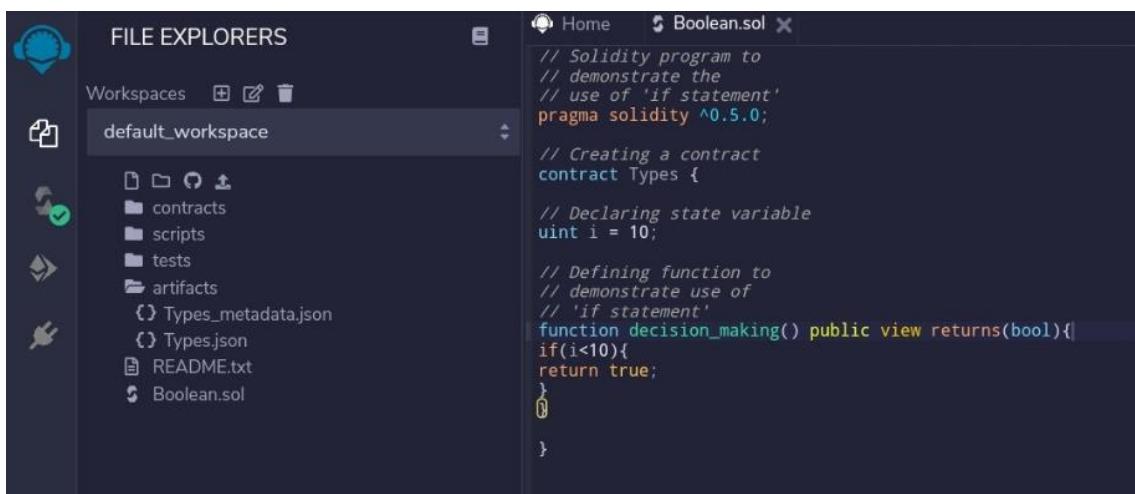
1 // Solidity contract to demonstrate Arithmetic Operator
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5 contract SolidityTest {
6
7 // Initializing variables
8 uint16 public a = 20;
9 uint16 public b = 10;
10 // Initializing a variable with sum
11 uint public sum = a + b;
12
13 // Initializing a variable with the difference
14 uint public diff = a - b;
15
16 // Initializing a variable with product
17 uint public mul = a * b;
18
19 // Initializing a variable with quotient
20 uint public div = a / b;
21
22 // Initializing a variable with modulus
23 uint public mod = a % b;
24
25 // Initializing a variable decrement value
26 uint public dec = --b;
27
28 // Initializing a variable with increment value
29 uint public inc = ++b;
30
31

```

(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 Visual Studio Code interface. On the left, the 'FILE EXPLORERS' sidebar displays a 'default_workspace' folder containing 'contracts', 'scripts', 'tests', 'artifacts' (with files 'Types_metadata.json' and 'Types.json'), 'README.txt', and 'Boolean.sol'. The 'Boolean.sol' file is open in the main editor area, showing the Solidity code provided in the previous text block.

```
// 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;  
        }  
    }  
}
```

The image shows the Solidity Compiler interface. On the left, there's a sidebar with icons for Home, Compiler, Languages, EVM Version, Compiler Configuration, Contracts, and Help. The main area is titled "SOLIDITY COMPILER".

COMPILER: 0.5.17+commit.d19bba13 (dropdown menu, checked for "Include nightly builds")

LANGUAGE: Solidity (dropdown menu)

EVM VERSION: compiler default (dropdown menu)

COMPILER CONFIGURATION:

- Auto compile
- Enable optimization (set to 200)
- Hide warnings

Contracts: Types (Boolean.sol) (dropdown menu)

Buttons:

- Compile Boolean.sol** (blue button)
- Publish on Swarm** (grey button)
- Publish on Ipfs** (grey button)
- Compilation Details** (grey button)

ABI **Bytecode** (links at the bottom)

Code Editor (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;
        }
    }
}
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, running transactions, and monitoring. The main area is divided into two sections: 'DEPLOY & RUN TRANSACTIONS' on the left and the 'Boolean.sol' contract details on the right.

DEPLOY & RUN TRANSACTIONS

- ENVIRONMENT:** JavaScript VM
- ACCOUNT:** 0x5B3...eddC4 (99.9999999)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** Types - Boolean.sol
- Buttons:** Deploy, Publish to IPFS

OR

- At Address:** Load contract from Address
- Transactions recorded:** 1
- Deployed Contracts:** TYPES AT 0XD91...39138 (MEMORY)
- Low level interactions:** decision_maki...
- CALLDATA:** Transact

Boolean.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;

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

```
// 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:

The screenshot shows the Solidity IDE interface. On the left is the 'FILE EXPLORERS' sidebar with a 'Workspaces' section containing 'default_workspace'. Inside 'default_workspace', there are folders for 'contracts', 'scripts', 'tests', and 'artifacts', along with files like 'Types_metadata.json', 'Types.json', 'test_metadata.json', 'test.json', 'README.txt', 'Boolean.sol', 'Book.sol', and 'If_else.sol'. The 'If_else.sol' file is currently selected and shown in the main code editor area.

```
// 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;
    }
}
```

The screenshot shows the 'SOLIDITY COMPILER' interface. On the left, under 'COMPILER', it's set to '0.5.17+commit.d19bba13'. Under 'LANGUAGE', it's set to 'Solidity'. Under 'EVM VERSION', it's set to 'compiler default'. Under 'COMPILER CONFIGURATION', there are options for 'Auto compile', 'Enable optimization' (set to 200), and 'Hide warnings'. A prominent blue button at the bottom left says 'Compile If_else.sol'. Below this, the 'CONTRACT' section shows 'Types (If_else.sol)' and three buttons: 'Publish on Swarm' (with a cloud icon), 'Publish on Ipfs' (with a green checkmark icon), and 'Compilation Details'. At the bottom, there are links for 'ABI' and '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;
    }
}
```

The screenshot shows the Truffle IDE interface divided into two main sections: the left panel for deployment and the right panel for code editing.

Left Panel (Deployment & Run Transactions):

- ENVIRONMENT:** JavaScript VM
- ACCOUNT:** 0x5B3...eddC4 (99.9999995)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** Types - If_else.sol
- Buttons:** Deploy, Publish to IPFS
- OR**: At Address, Load contract from Address
- Transactions recorded (7):** Deployed Contracts
- Low level interactions:** CALLDATA, Transact

Right Panel (Code Editor):

```
// 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;
    }
}
```

(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. On the left, the "SOLIDITY COMPILER" sidebar includes settings for the compiler version (0.4.26+commit.4563c3fc), language (Solidity), EVM version (compiler default), and compilation options (Auto compile, Enable optimization). A prominent blue button at the bottom left says "Compile string.sol". Below this, the "CONTRACT" section shows "Test (string.sol)" selected. Underneath are buttons for "Publish on Swarm" and "Publish on Ipfs". At the bottom of the sidebar is a "Compilation Details" button. The main area displays the Solidity source code:

```
// 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;
    }
}
```

Below the code, the deployment interface shows fields for "from" (0x58380a6a701c568545dCfC803Fc875F56bedd84), "to" (SolidityTest.(constructor)), and "gas" (3000000 gas). A search bar at the top right is labeled "Search with transaction hash or address".

The screenshot shows the "DEPLOY & RUN TRANSACTIONS" interface. On the left, the "SOLIDITY_VAR_TEST AT 0X358...D5EE3" section has a "state_var" tab open, showing a "Low level interactions" field with "CALldata" set to "0" and a "Transact" button. Below it are sections for "TEST AT 0X9D7...B5E99 (MEMORY)" and "SOLIDITYTEST AT 0XD2A...FD005 (MEM)". The "str_out" section also has a "Low level interactions" field with "CALldata" set to "0" and a "Transact" button. The main area displays the same Solidity source code as the previous screenshot. The deployment interface at the bottom is identical to the one in the first screenshot.

Practical 5

Implement and demonstrate the use of the following in Solidity :

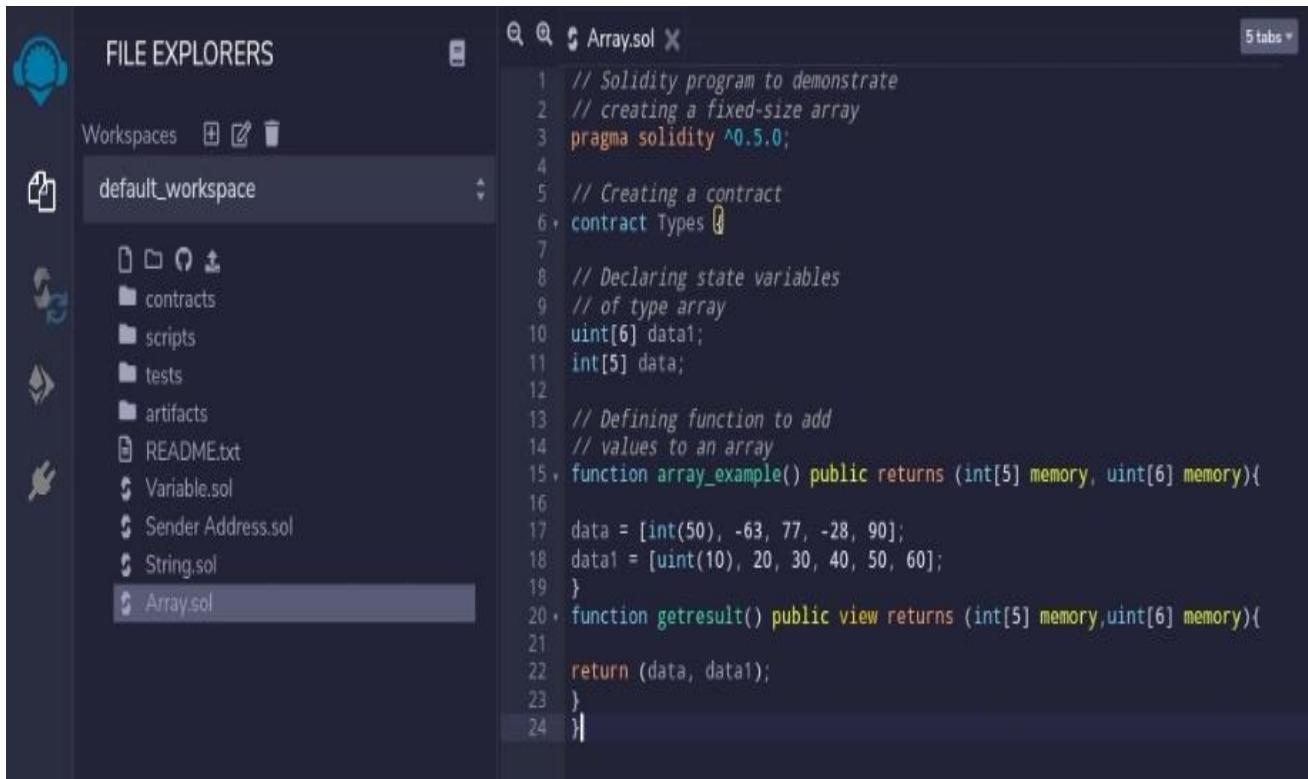
- (I).Arrays
- (II).Enums
- (III).Structs
- (IV).Mappings
- (V).Coversations
- (VI).Ether Units
- (VII).Special Varaibles

(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);  
}  
}
```

Output:



The screenshot shows the Solidity IDE interface. On the left, the 'FILE EXPLORERS' panel displays the workspace structure under 'default_workspace'. It includes folders for contracts, scripts, tests, artifacts, and files like README.txt, Variable.sol, SenderAddress.sol, String.sol, and Array.sol. The 'Array.sol' file is currently selected. On the right, the main code editor window shows the Solidity code for 'Array.sol'. The code defines a contract with state variables, a function to add values to an array, and a function to get the result.

```
// 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 arrayExample() 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);  
    }  
}
```

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons and settings:

- SOLIDITY COMPILER**
- COMPILER:** 0.5.17+commitd19bba13
- Include nightly builds
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION**
- Auto compile
- Enable optimization (set to 200)
- Hide warnings

Below these settings is a large blue button labeled **Compile Array.sol**.

On the right, the code editor displays the **Array.sol** file:

```
// 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] data;
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);
}
```

At the bottom of the code editor, there are three buttons:

- Publish on Swarm** (with a Swarm icon)
- Publish on Ipfs** (with an IPFS icon)
- Compilation Details**

Below the buttons are links for ABI and Bytecode.

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, running, and monitoring transactions. The main area is divided into sections:

- ENVIRONMENT:** Set to "JavaScript VM".
- ACCOUNT:** Address 0x5B3...eddC4 (99.9999999).
- GAS LIMIT:** Set to 3000000.
- VALUE:** 0 wei.
- CONTRACT:** Selected "Types - Array.sol".
- Deploy:** Orange button.
- Publish to IPFS:** Gray button.
- OR**
- At Address:** "Load contract from Address".
- Transactions recorded:** 3.
- Deployed Contracts:** A list showing "TYPES AT 0xD91...39138 (MEMORY)" with a deployed contract named "array_example".
- array_example:** Shows two functions: "getresult" and "array_example".
- getresult:** Returns "0: int256(5): 50,-63,77,-28,90" and "1: uint256[6]: 10,20,30,40,50,60".
- Low level interactions:** CALDATA section with a "Transact" button.

On the right, the code editor displays the Solidity source code for "Array.sol":

```

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
17 data = [int(50), -63, 77, -28, 90];
18 data1 = [uint(10), 20, 30, 40, 50, 60];
19 }
20 function getresult() public view returns (int[5] memory,uint[6] memory){
21
22 return (data, data1);
23 }
24 }

```

(III).Structs

```
pragma solidity ^0.5.0;
```

```
contract test {
```

```
    struct Book {
```

```
        string title;
```

```
        string author;
```

```
        uint book_id;
```

```
}
```

```
Book book;
```

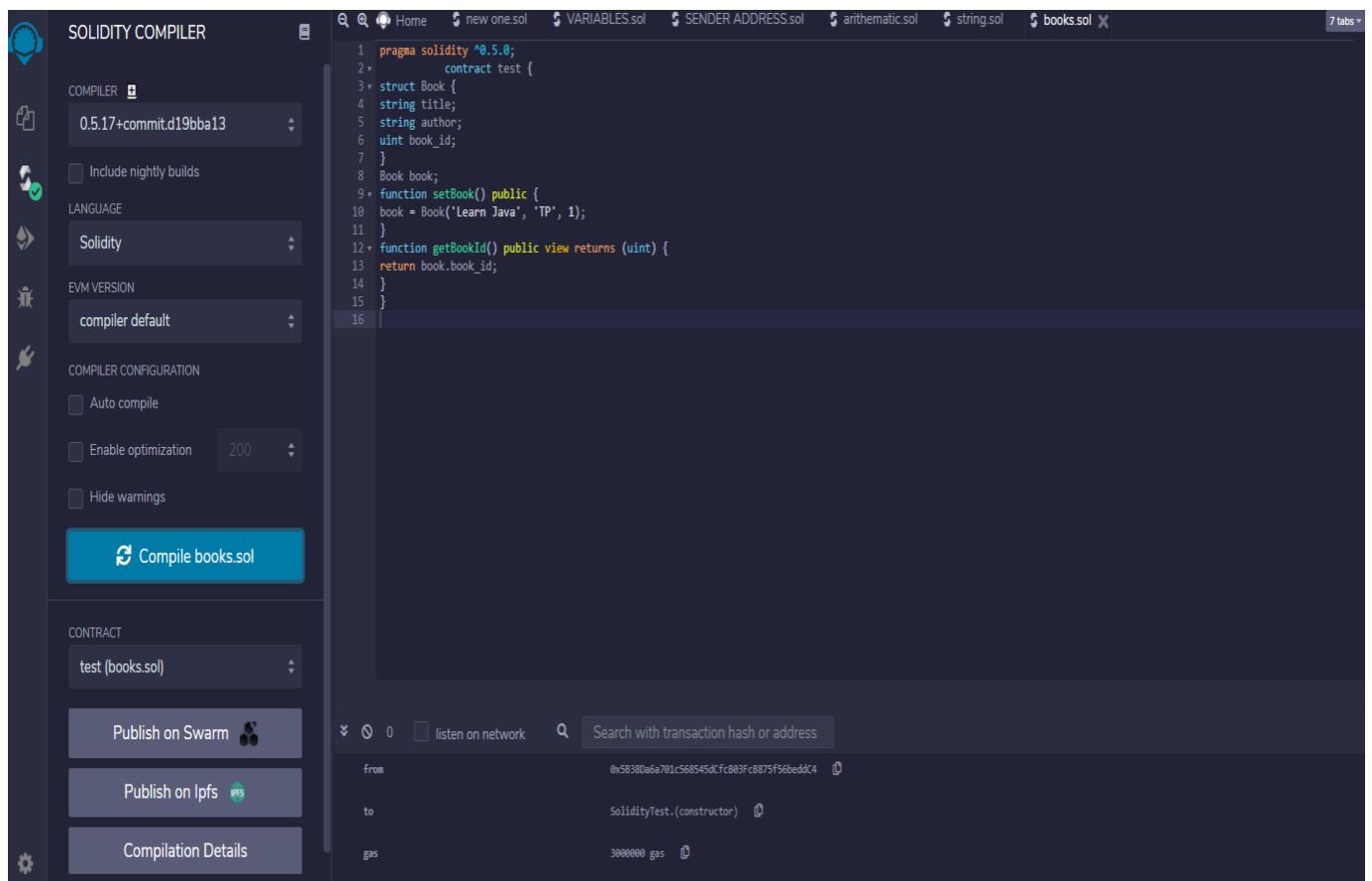
```

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 with the following details:

- Compiler:** 0.5.17+commit.d19bba13
- Include nightly builds:** Unchecked
- Language:** Solidity
- EVM Version:** compiler default
- Compiler Configuration:**
 - Auto compile: Unchecked
 - Enable optimization: 200
 - Hide warnings: Unchecked
- Contract:** test (books.sol)
- Buttons:**
 - Publish on Swarm
 - Publish on Ipfs
 - Compilation Details
- Output Area:** Displays the Solidity source code with line numbers 1 through 16.
- Bottom Panel:** Shows transaction details: from (0x58380a6a701c568545dFc003FcB875f56beddC4), to (SolidityTest.(constructor)), and gas (3000000).

The screenshot shows the Truffle IDE interface. On the left, there's a sidebar with icons for file operations like new, open, save, and delete. Below that is a section for 'DEPLOY & RUN TRANSACTIONS' which lists several test environments: 'TEST AT 0x9D7...B5E99 (MEMORY)', 'SOLIDITYTEST AT 0xD2A...FD005 (MEMORY)', 'TEST AT 0xDDA...5482D (MEMORY)', and 'TEST AT 0x0FC...9A836 (MEMORY)'. Each environment has a 'str_out' button. Under 'TEST AT 0xDDA...5482D (MEMORY)', there are buttons for 'setBook' and 'getBookId'. The right side of the interface displays the Solidity code for a 'test' contract:

```

1 pragma solidity ^0.5.0;
2 contract test {
3     struct Book {
4         string title;
5         string author;
6         uint book_id;
7     }
8     Book book;
9     function setBook() public {
10        book = Book('Learn Java', 'IP', 1);
11    }
12    function getBookId() public view returns (uint) {
13        return book.book_id;
14    }
15 }
16

```

Below the code, there's a 'Low level interactions' section with a 'CALLDATA' tab and a 'Transact' button. At the bottom, there's a transaction details panel with fields for 'from', 'to', 'gas', and 'transaction cost', along with a 'listen on network' button and a search bar.

(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:

The image shows the Solidity Compiler interface. On the left, there's a sidebar with various icons: a blue headphones icon, a white square with a blue border, a green circular icon with a checkmark, a yellow lightning bolt, and a hand icon. The main area is titled "SOLIDITY COMPILER". It includes sections for "COMPILER" (set to 0.5.17+commit.d19bba13), "LANGUAGE" (set to Solidity), "EVM VERSION" (set to compiler default), and "COMPILER CONFIGURATION" with options for Auto compile, Enable optimization (set to 200), and Hide warnings. A prominent blue button labeled "Compile Mapping.sol" is centered below these settings. To the right, there's a code editor window titled "Function_program.sol" and "Mapping.sol". The "Mapping.sol" file contains the following Solidity code:

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

The screenshot shows the Truffle UI interface divided into two main sections: "DEPLOY & RUN TRANSACTIONS" on the left and a code editor on the right.

Left Panel: DEPLOY & RUN TRANSACTIONS

- ENVIRONMENT:** JavaScript VM
- ACCOUNT:** 0x5B3...eddC4 (99.999999ε)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** LedgerBalance - Mapping.sol
- Buttons:** Deploy, Publish to IPFS
- OR:** At Address, Load contract from Address
- Transactions recorded (15):**
 - TYPES AT 0xD91...39138 (MEMORY)
 - TEST AT 0xDBB...33FA8 (MEMORY)
 - TYPES AT 0xDA0...42B53 (MEMORY)
 - TEST AT 0x9D7...B5E99 (MEMORY)
 - LEDGERBALANCE AT 0xD2A...FD005 (!)
- Function Buttons:** updateBalance
- Low level interactions:** CALldata, Transact

Right Panel: Code Editor

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

```

pragma solidity ^0.5.0;

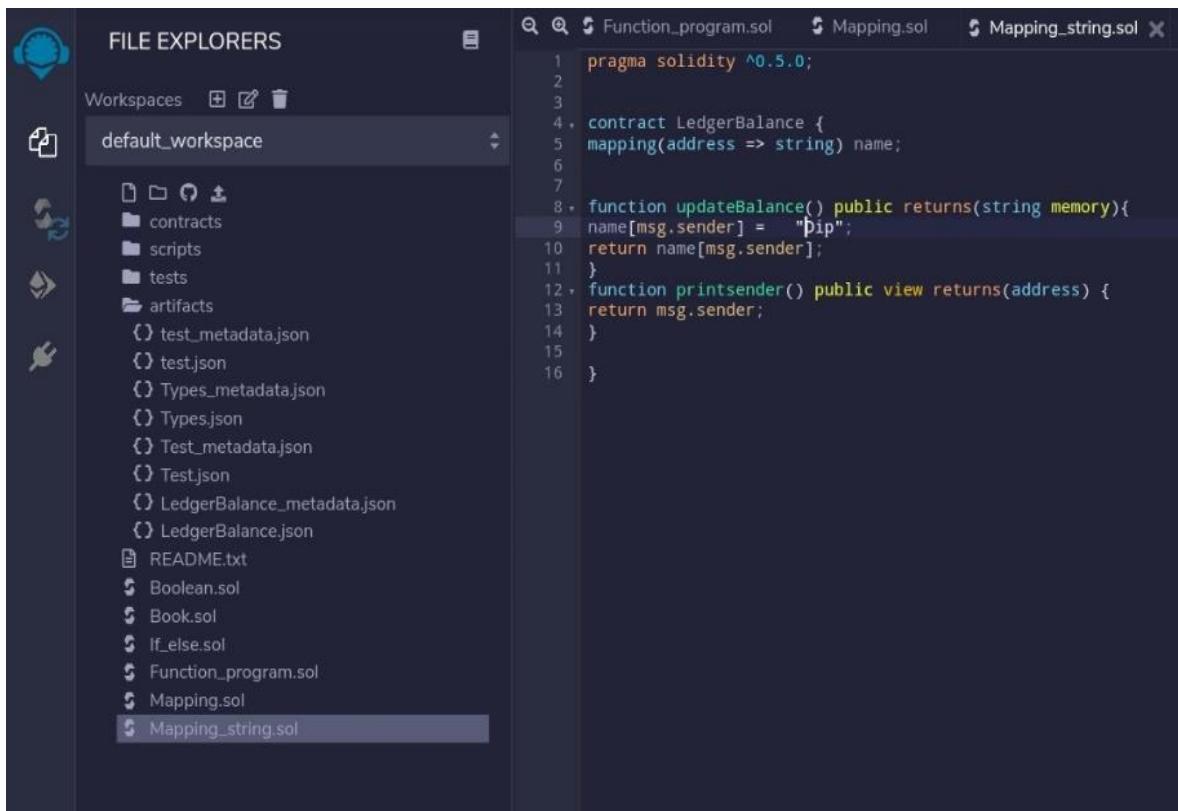
contract LedgerBalance {
    mapping(address => string) name;

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

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

```

Output:

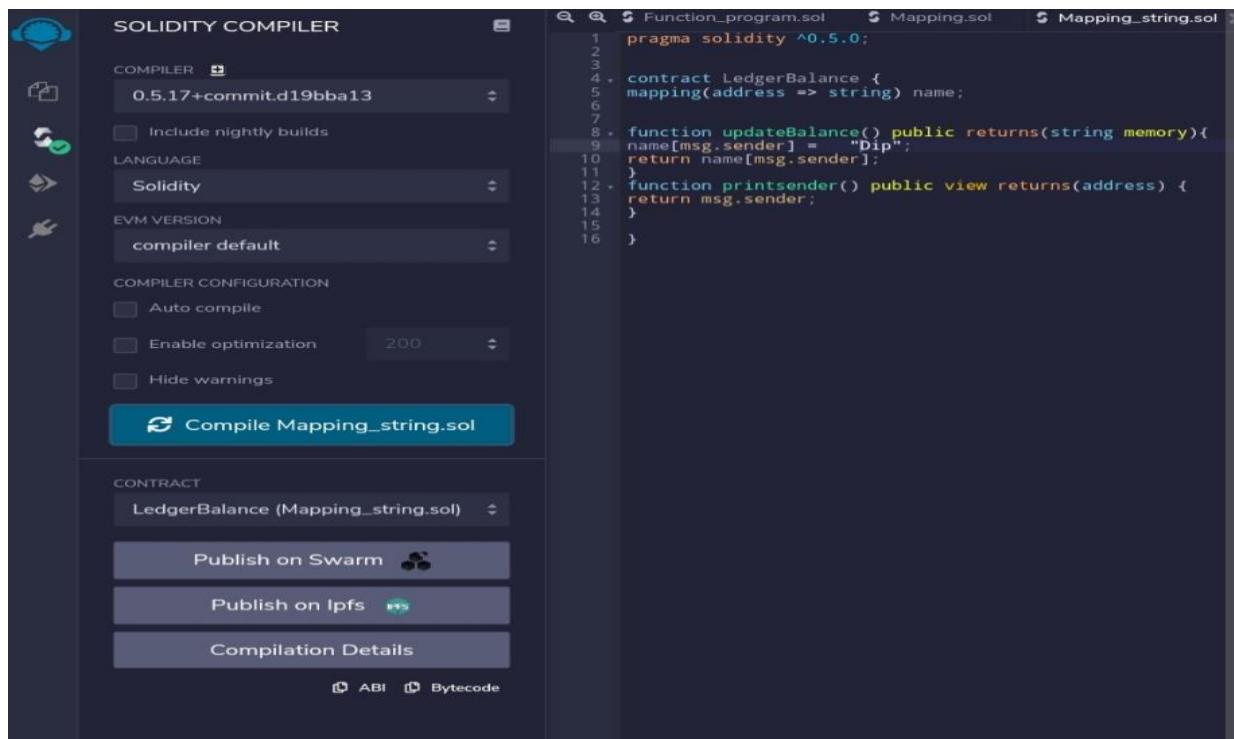


The screenshot shows the Solidity IDE interface. On the left is the 'FILE EXPLORERS' sidebar, which displays the current workspace ('default_workspace') and its contents: contracts, scripts, tests, artifacts, and various JSON files like test_metadata.json, test.json, and LedgerBalance.json. In the center is the code editor window, which contains the Solidity code for the 'LedgerBalance' contract. The code defines a mapping from addresses to strings, includes an 'updateBalance' function that sets the value to 'Dip', and a 'printsender' function that returns the sender's address.

```

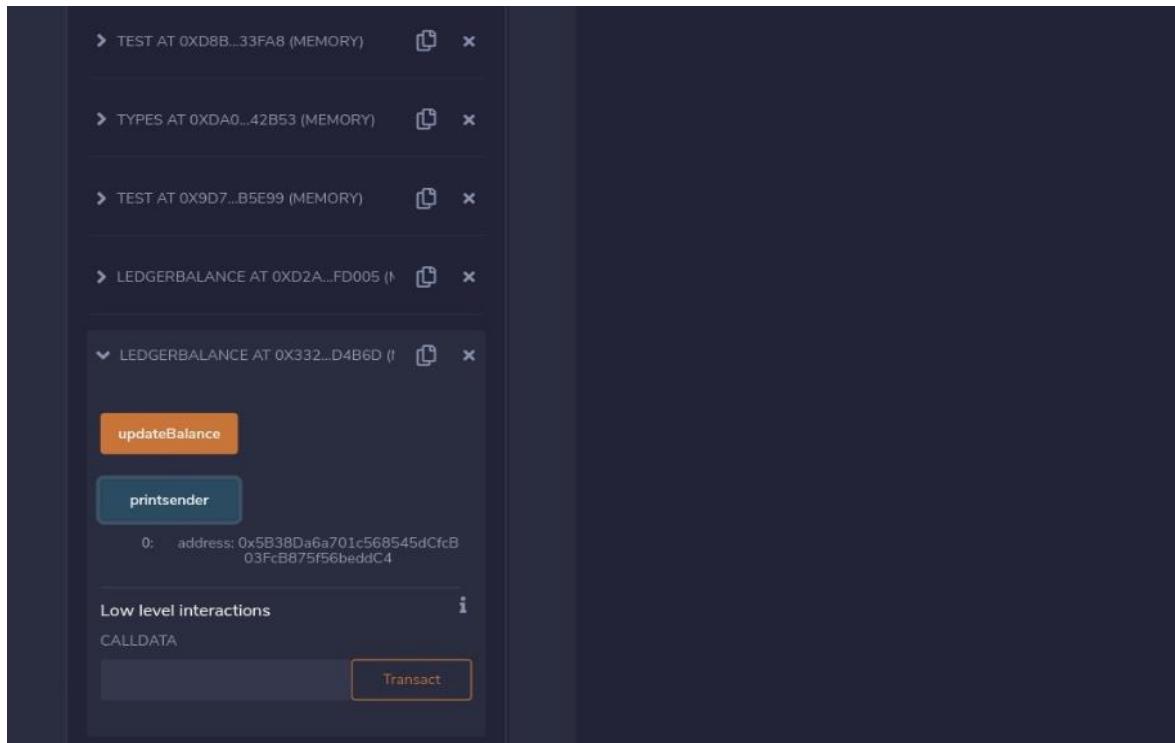
1 pragma solidity ^0.5.0;
2
3
4 contract LedgerBalance {
5     mapping(address => string) name;
6
7
8     function updateBalance() public returns(string memory){
9         name[msg.sender] = "Dip";
10    return name[msg.sender];
11 }
12 function printsender() public view returns(address) {
13    return msg.sender;
14 }
15
16 }

```



The screenshot shows the Solidity Compiler interface. On the left, there are configuration options 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 labeled "Compile Mapping_string.sol" is visible. On the right, the source code for `Mapping_string.sol` is displayed:

```
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 contract interaction interface for the `LedgerBalance` contract. It lists several test results and interactions:

- TEST AT 0XD8B...33FA8 (MEMORY)
- TYPES AT 0XDA0...42B53 (MEMORY)
- TEST AT 0X9D7...B5E99 (MEMORY)
- LEDGERBALANCE AT 0XD2A...FD005 (I)
- LEDGERBALANCE AT 0X332...D4B6D (I)

Under the last interaction, there are two buttons: `updateBalance` (orange) and `printsender`. The `printsender` button has a tooltip indicating it returns the address: 0x5B38Da6a701c568545dCfcB 03FcB875f56beddC4. Below this, there is a section for "Low level interactions" and a "CALLDATA" field with a "Transact" button.

Practical 6

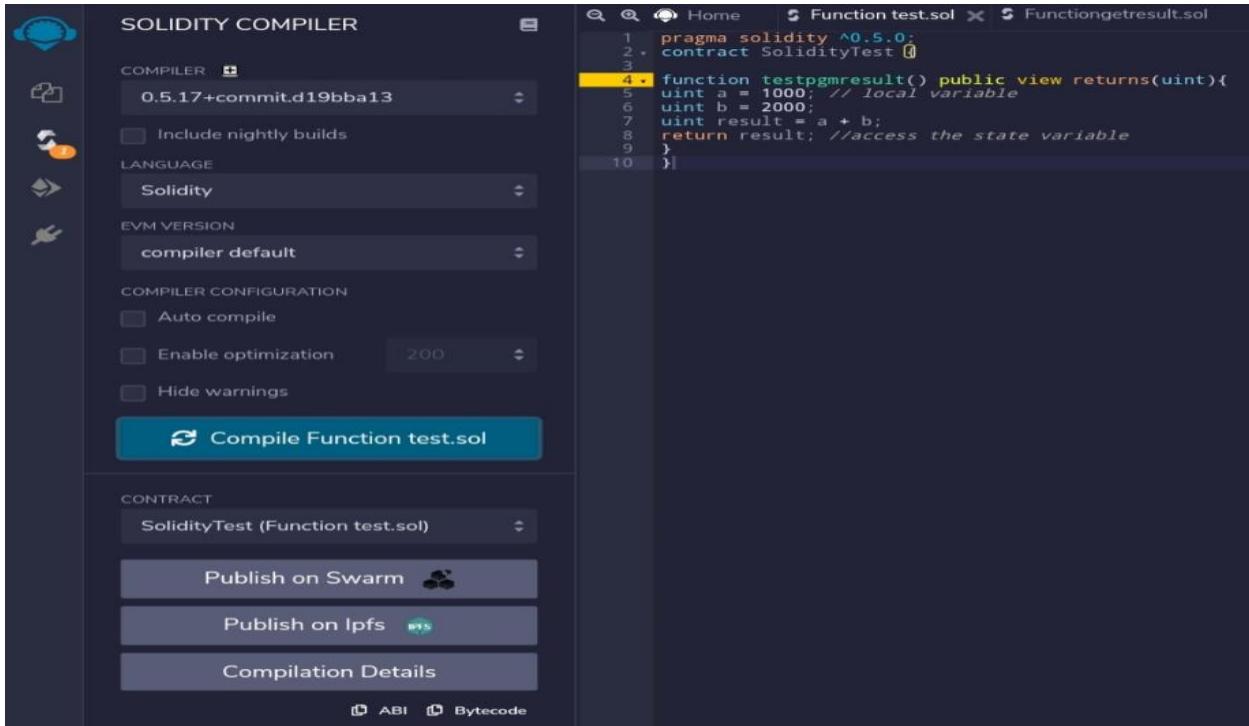
Implement and demonstrate the use of the following in Solidity :

- (I).Functions
- (II).View Functions
- (III).Pure Functions
- (IV).Fallback Functions
- (V).Function Overloading
- (VI).Mathematical Functions
- (VII).Cryptographic Functions

(I).Functions

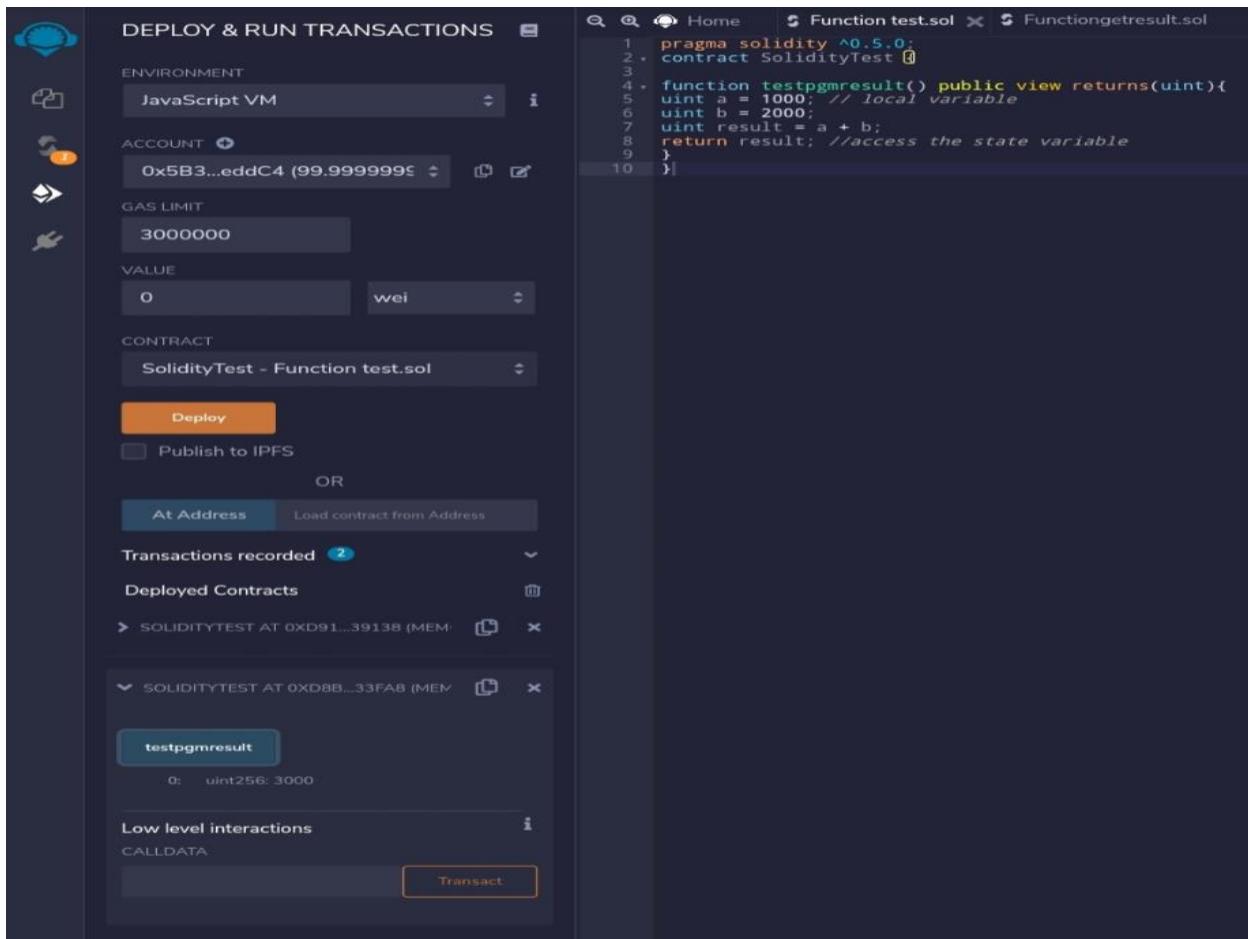
```
pragma solidity ^0.5.0;  
  
contract SolidityTest {  
  
    function testpgmresult() public view returns(uint){  
        uint a = 1000; // local variable  
  
        uint b = 2000;  
  
        uint result = a + b;  
  
        return result; //access the state variable  
    }  
}
```

Output:



The screenshot shows the Solidity Compiler interface. On the left, 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 like Auto compile, Enable optimization (set to 200), and Hide warnings. A prominent blue button labeled "Compile Function test.sol" is centered. Below it, under "CONTRACT", it says "SolidityTest (Function test.sol)". There are three buttons: "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.

```
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
    }
}
```



(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;

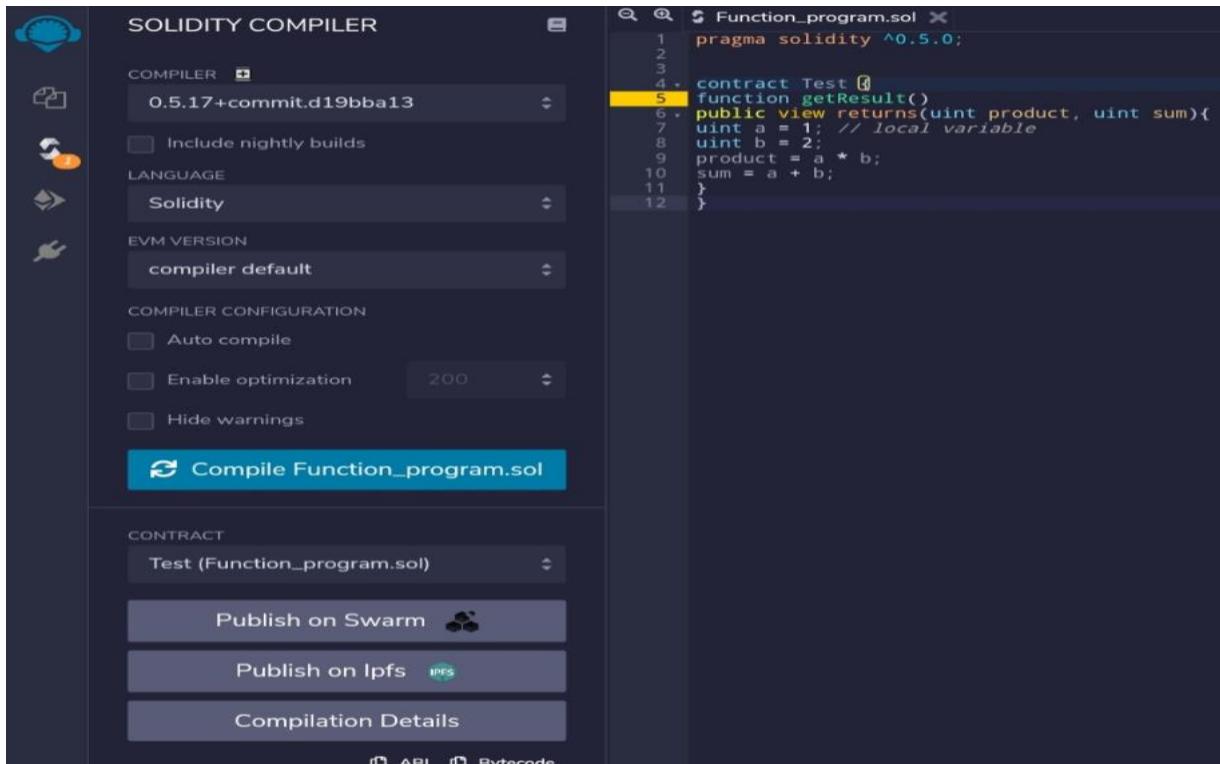
product = a * b;

sum = a + b;

}

}
```

Output:



The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with icons for code, terminal, and help. The main area has tabs for 'SOLIDITY COMPILER' (selected), 'TESTER', and 'CONTRACTOR'. Under 'SOLIDITY COMPILER', settings include 'COMPILER' (0.5.17+commit.d19bba13), 'LANGUAGE' (Solidity), 'EVM VERSION' (compiler default), and 'COMPILER CONFIGURATION' with options like 'Auto compile', 'Enable optimization' (set to 200), and 'Hide warnings'. A large blue button labeled 'Compile Function_program.sol' is prominent. Below it, under 'CONTRACT', the contract 'Test (Function_program.sol)' is selected, with options to 'Publish on Swarm' (with a Swarm icon) and 'Publish on Ipfs' (with an IPFS icon). At the bottom, there are links for 'Compilation Details', 'ABI', and 'Bytecode'.

```
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;
    }
}
```

The screenshot shows the Truffle UI interface for deploying a Solidity contract. On the left, there's a sidebar with icons for account management, file operations, and network selection. The main area is titled "DEPLOY & RUN TRANSACTIONS". It includes fields for "ENVIRONMENT" (set to "JavaScript VM"), "ACCOUNT" (selected account address), "GAS LIMIT" (set to 3000000), "VALUE" (set to 0 wei), and a "CONTRACT" dropdown set to "Test - Function_program.sol". A prominent orange "Deploy" button is centered below these fields. Below the "Deploy" button is a checkbox for "Publish to IPFS". The interface then branches into two sections: "At Address" (selected) and "Load contract from Address". Under "At Address", it shows "Transactions recorded" (8) and a list of "Deployed Contracts" with addresses like 0xD91...3913B, 0xD8B...33FA8, 0xDA0...42B53, and 0x9D7...B5E99. Each contract entry has a "View" icon and a "Delete" icon. In the bottom right corner of the deployed contracts list, there is a "getResult" button, which is highlighted with a blue border. Below this button, the output of the function call is shown: "0: uint256: product 2" and "1: uint256: sum 3".

(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

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:

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

LANGUAGE: Solidity

EVM VERSION: compiler default

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

Compile calling function external.sol

CONTRACT: C (calling function external.sol)

Publish on Swarm, Publish on ipfs

Compilation Details

DEPLOY & RUN TRANSACTIONS

TEST AT 0x0FC...9A836 (MEMORY)

- setBook
- getBookId

Low level interactions

CALLEDATA: **Transact**

C AT 0XAED...96B8B (MEMORY)

- updateData
- getData
- info

Transact

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(); }
}

from: 0x58380a6a701c568545dCfc803Fc8875f56beddC4

to: SolidityTest.(constructor)

gas: 3000000 gas

Search with transaction hash or address

(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);
}
}
```

Output:

The screenshot shows the Truffle UI interface. On the left, the 'SOLIDITY COMPILER' panel is open, displaying the code for `overloading.sol`. The code defines a contract with three functions: `getSum` (with two or three arguments), `callSumWithTwoArguments`, and `callSumWithThreeArguments`. The compiler version is set to 0.5.17+commit.d19bba13, and the language is Solidity. The right side of the interface shows the 'CONTRACT' panel with the deployed contract details: address 0x5B380a6a701c568545dCfc803fc8875f56bedd4, constructor call, and gas limit of 3000000.

```
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);
    }
}
```

```

1 pragma solidity ^0.5.0;
2
3 contract Test {
4     function getSum(uint a, uint b) public pure returns(uint){
5         return a + b;
6     }
7     function getSum(uint a, uint b, uint c) public pure returns(uint){
8         return a + b + c;
9     }
10    function callSumWithTwoArguments() public pure returns(uint){
11        return getSum(1,2);
12    }
13    function callSumWithThreeArguments() public pure returns(uint){
14        return getSum(1,2,3);
15    }
16

```

(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 Solidity Compiler interface shows the following configuration:

- Compiler: 0.5.17+commit.d19bba13
- Include nightly builds: Unchecked
- Language: Solidity
- EVM Version: compiler default
- Compiler Configuration:
 - Auto compile: Unchecked
 - Enable optimization: Checked (value 200)
 - Hide warnings: Unchecked
- Contract: Test (Calladdmod.sol)
- Buttons:
 - Publish on Swarm
 - Publish on Ipfs
 - Compilation Details
- ABI and Bytecode links

A blue button at the bottom left says "Compile Calladdmod.sol".

```
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);
    }
}
```

The deployment interface shows the following details:

- Contract: Test - Calladdmod.sol
- Deploy button (highlighted in orange)
- Publish to IPFS checkbox
- OR
- At Address: Load contract from Address
- Transactions recorded: 2
- Deployed Contracts:
 - TEST AT 0xD91...39138 (MEMORY)
 - TEST AT 0xD8B...33FAB (MEMORY)
 - callAddMod
 - 0: uint256: 0
 - callMulMod
 - 0: uint256: 2

Low level interactions: CALldata

Transact button

(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:

The screenshot shows the Truffle UI interface. On the left, the 'SOLIDITY COMPILER' sidebar is visible, containing 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 blue button labeled 'Compile cryptodata.sol' is present. Below this, the 'CONTRACT' section shows 'Test (cryptodata.sol)' selected, with options to 'Publish on Swarm' or 'Publish on Ipfs'. On the right, the main workspace displays two tabs: 'overloading.sol' and 'cryptodata.sol'. The 'cryptodata.sol' tab contains the Solidity code provided above. Below the tabs, a transaction details panel is shown, with fields for 'from' (0x5830de6a701c568545dFc880Fc8875f56bed0C4), 'to' (SolidityTest.(constructor)), 'gas' (3000000 gas), and 'transaction cost' (116000 gas).

```
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 }
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons. The main area has tabs for 'overloading.sol' and 'cryptodata.sol'. The 'overloading.sol' tab contains the following 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 }

```

Below the code, there's a 'call' button and a result field showing '0: uint256: 9'. Under 'Low level interactions', there's a 'CALLDATA' section with a 'Transact' button.

The 'cryptodata.sol' tab is also visible. A modal window titled 'TEST AT 0x332...D4B6D (MEMORY)' shows two results for different hash functions:

- callkeccak256**: Result: bytes32: 0x2c96055cb975b6296
46e25b4c09bf530fb8fffa92358fb679b
ceb0e5538eed8
- callsha256**: Result: bytes32: 0xe24dd2210803ba737a
9bd9e3163a4ca87b63201c3bc32b68fb
122ca52efff36

Under 'Low level interactions', there's another 'CALLDATA' section with a 'Transact' button. At the bottom, there's a transaction input form with fields for 'from', 'to', and '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;
```

```
}
```

```
// Defining function to
```

```
// print the sum of
```

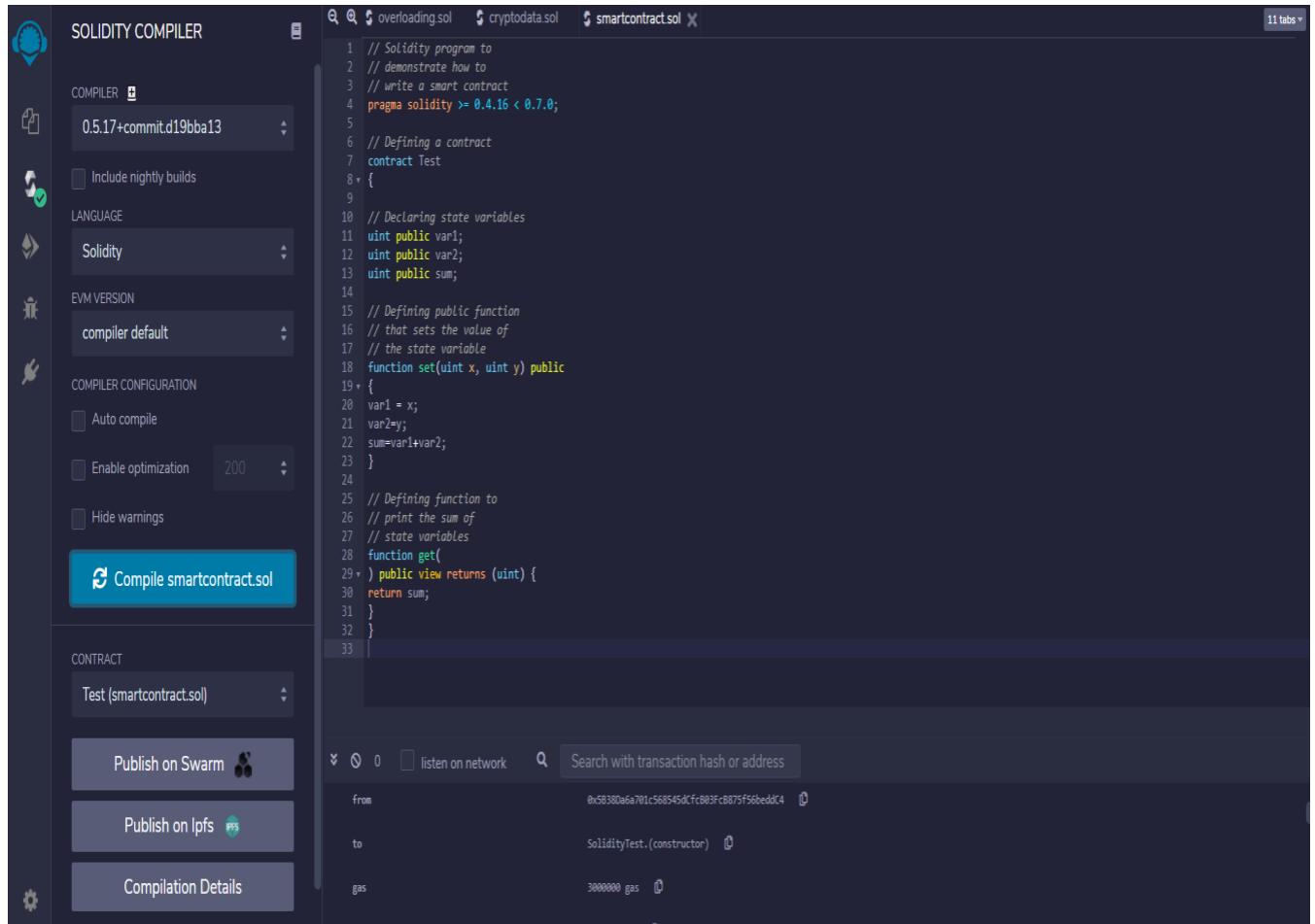
```
// state variables
```

```
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 icons and settings. The 'COMPILER' dropdown is set to '0.5.17+commit.d19bba13'. Under 'LANGUAGE', 'Solidity' is selected. 'EVM VERSION' is set to 'compiler default'. In the 'COMPILER CONFIGURATION' section, 'Auto compile' is checked, 'Enable optimization' is set to 200, and 'Hide warnings' is checked. A large blue button labeled 'Compile smartcontract.sol' is prominent. Below it, the 'CONTRACT' dropdown is set to 'Test (smartcontract.sol)'. There are three buttons: 'Publish on Swarm' (with a Swarm icon), 'Publish on Ipfs' (with an IPFS icon), and 'Compilation Details'. The main area is a code editor with tabs for 'overloading.sol', 'cryptodata.sol', and 'smartcontract.sol'. The 'smartcontract.sol' tab is active, displaying the following Solidity code:

```
// 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;  
    }  
}
```

Below the code editor, there are fields for 'from' (set to '0x58380a6a701c568545dcfc803fc8875f56beddC4'), 'to' (set to 'SolidityTest.(constructor)'), and 'gas' (set to '3000000 gas'). A search bar at the bottom says 'Search with transaction hash or address'.

```

// 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

get

sum

var1

var2

Low level interactions

0 bytes32 result 0xe24dd2210803b4737a 9bd9e31634a4ca80763201c3bc32b68fb 122ca52eff36

Transact

transaction hash 0x85fcab5de9819161eb66c32b59aab8e5c427b740d75c05a17394c2854b3143c6

Search with transaction hash or address

Debug

(II).Inheritance

```
// Solidity program to demonstrate Single Inheritance
```

```
pragma solidity >=0.4.22 <0.6.0;
```

```
// Defining contract
```

```
contract parent{
```

```
// Declaring internal state variable
```

```
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;
}

}

// 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:

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.99999999999999)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: child - Hierarchical.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded: 26

Deployed Contracts:

- ▶ TYPES AT 0XD91...39138 (MEMORY)
- ▶ TEST AT 0XDBB...33FAB (MEMORY)
- ▶ TYPES AT 0XDA0...42B53 (MEMORY)
- ▶ TEST AT 0X9D7...B5E99 (MEMORY)
- ▶ LEDGERBALANCE AT 0XD2A...FD005 (MEMORY)
- ▶ LEDGERBALANCE AT 0X332...D4B6D (MEMORY)
- ▼ CHILD AT 0X406...2CFBC (MEMORY)

setValue

getValue

o: uint256: 30

Low level interactions

CALLDATA

Transact

Hierarchical.sol

```
1 // Solidity program to
2 // demonstrate
3 // Single Inheritance
4 pragma solidity >=0.4.22 <0.6.0;
5
6
7 // Defining contract
8 contract parent{
9
10 // Declaring internal
11 // state variable
12 uint internal sum;
13
14 // Defining external function
15 // to set value of internal
16 // state variable sum
17 function setValue() external {
18     uint a = 10;
19     uint b = 20;
20     sum = a + b;
21 }
22 }
23
24
25 // Defining child contract
26 contract child is parent{
27
28 // Defining external function
29 // to return value of
30 // internal state variable sum
31 function getValue() external view returns(uint) {
32     return sum;
33 }
34 }
35 }
```

(III).Constructors

```
// Solidity program to demonstrate  
// creating a constructor  
pragma solidity ^0.5.0;
```

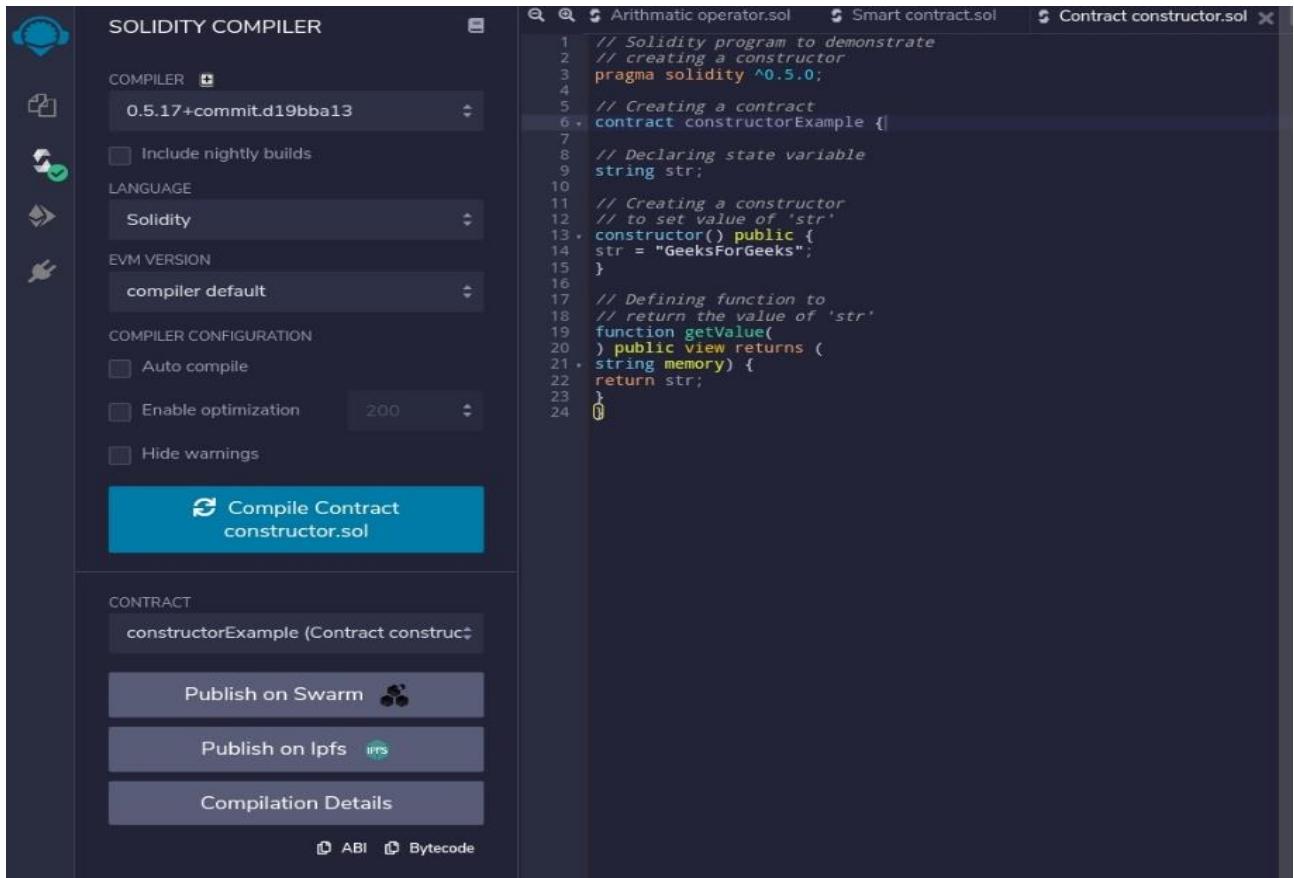
```
// Creating a contract  
contract constructorExample {
```

```
// 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. On the left, there's a sidebar with various icons. The main area is titled "SOLIDITY COMPILER". It includes sections for "COMPILER" (version 0.5.17+commit.d19bba13), "LANGUAGE" (Solidity), "EVM VERSION" (compiler default), and "COMPILER CONFIGURATION" (Auto compile, Enable optimization set to 200). A large blue button at the bottom says "Compile Contract constructor.sol". On the right, there are tabs for "Arithmetic operator.sol", "Smart contract.sol", and "Contract constructor.sol". The "Contract constructor.sol" tab is active, displaying the following Solidity code:

```
// 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;
    }
}
```

The screenshot shows the Truffle UI interface for deploying and running transactions. The left sidebar contains environment settings (JavaScript VM, Account: 0x5B3...eddC4, Gas Limit: 3000000), value input (0 wei), and a contract selection dropdown for 'constructorExample'. A prominent orange 'Deploy' button is visible. Below it is a checkbox for 'Publish to IPFS'. The main area displays a list of deployed contracts and their transaction details. The 'Deployed Contracts' section lists four contracts: 'SOLIDITYTEST' at address 0xD91..., 'SOLIDITYTEST' at 0xD8B..., 'SOLIDITYTEST' at 0xF8E..., and 'TEST' at 0xD7A... . Below this is a expanded section for 'CONSTRUCTOREXAMPLE' at address 0xDA0... . Underneath, a blue button labeled 'getValue' is shown, along with the response '0: string: GeeksForGeeks'. At the bottom, there's a 'Low level interactions' section with a 'Transact' button.

```
// 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;
    }
}
```

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){
```

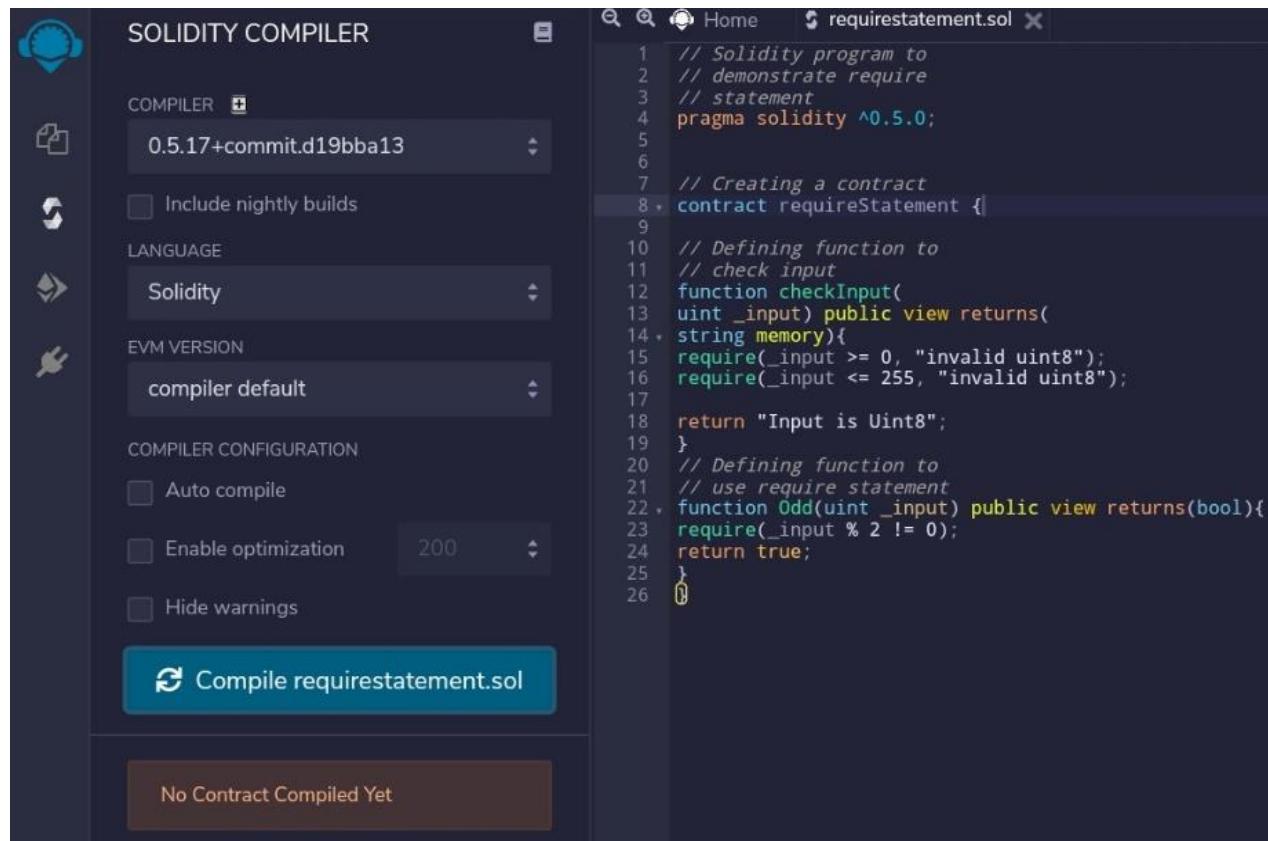
```
require(_input % 2 != 0);
```

```
return true;
```

```
}
```

}

Output:



The image shows the Solidity Compiler interface. On the left, there are several icons: a blue gear, a white square with a blue border, a blue gear with a white square, a blue gear with a white gear, and a blue gear with a white hand. The main area has a dark background with light-colored text. At the top, it says "SOLIDITY COMPILER". Below that is a "COMPILER" dropdown set to "0.5.17+commit.d19bba13", a checkbox for "Include nightly builds" which is unchecked, and a "LANGUAGE" dropdown set to "Solidity". Under "EVM VERSION", it says "compiler default". In the "COMPILER CONFIGURATION" section, there are three checkboxes: "Auto compile" (unchecked), "Enable optimization" (unchecked with a dropdown menu showing "200"), and "Hide warnings" (unchecked). At the bottom of this section is a large blue button with a circular arrow icon and the text "Compile requirestatement.sol". Below this button is a brown rectangular bar with the text "No Contract Compiled Yet". On the right side of the interface, there is a file browser window titled "requirestatement.sol" showing the following Solidity code:

```
// 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;
    }
}
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for Ethereum, Truffle, and Ganache. The main area has tabs for "DEPLOY & RUN TRANSACTIONS" and "requireStatement.sol". Under "DEPLOY & RUN TRANSACTIONS", the "ENVIRONMENT" is set to "JavaScript VM", "ACCOUNT" is "0x5B3...eddC4 (99.9999999)", "GAS LIMIT" is "3000000", and "VALUE" is "0 wei". Below these are sections for "CONTRACT" (set to "requireStatement - requirestatement.sol") and buttons for "Deploy" and "Publish to IPFS". There's also an "OR" section with "At Address" selected and "Load contract from Address". Under "Transactions recorded", there's one entry for "REQUIRESTATEMENT AT 0xD91...3913". The right side shows the Solidity code for the "requireStatement" contract, which includes functions "checkInput" and "Odd". The "checkInput" function returns "Input is UInt8" for input 254. The "Odd" function returns "true" for input 253.

```
// 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;
    }
}

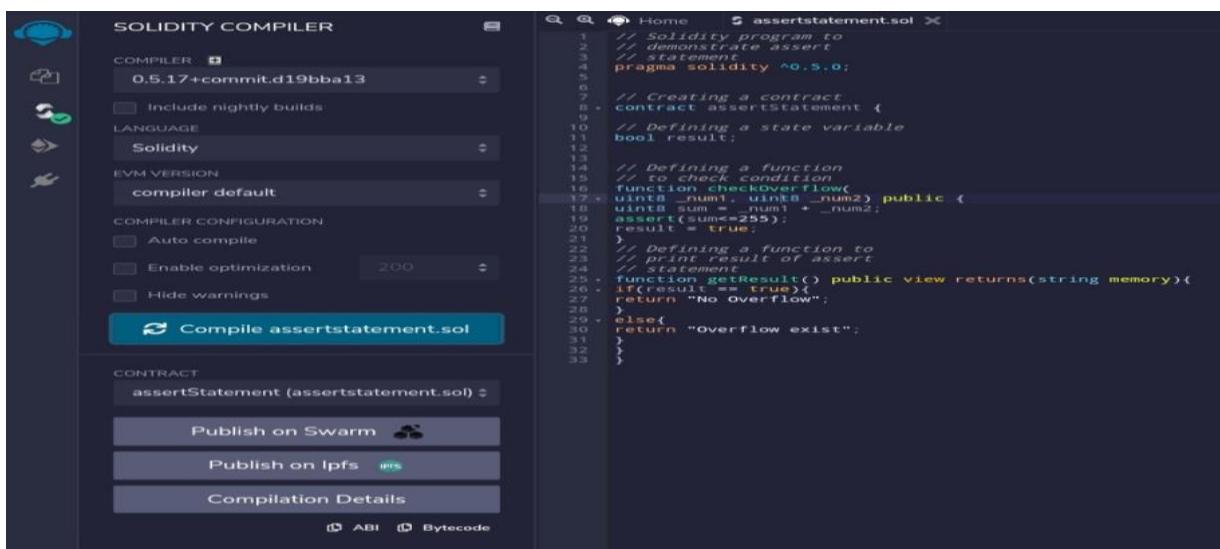
```

```

// 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:



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

- 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:
- Contract** section: assertStatement (assertstatement.sol)
- Code Area** (right side):


```

1 // Solidity program to
2 // demonstrate assert
3 // statement
4 pragma solidity ^0.5.0;
5
6 // Creating a contract
7 contract assertStatement {
8
9     // Defining a state variable
10    bool result;
11
12
13
14    // Defining a function
15    // To check condition
16    function checkOverflow(
17        uint8 _num1, uint8 _num2) public {
18        uint8 sum = _num1 + _num2;
19        assert(sum<=255);
20        result = true;
21    }
22    // Defining a function to
23    // print result of assert
24    // statement
25    function getResult() public view returns(string memory){
26        if(result==true){
27            return "No Overflow";
28        }
29        else{
30            return "Overflow exist";
31        }
32    }
33 }
```
- Buttons at the bottom:**
 - Compile assertstatement.sol
 - Publish on Swarm
 - Publish on Ipfs
 - Compilation Details
 - ABI
 - Bytecode

```

// 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 {
        uint8 _num1, uint8 _num2;
        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";
        }
    }
}

```

// 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{

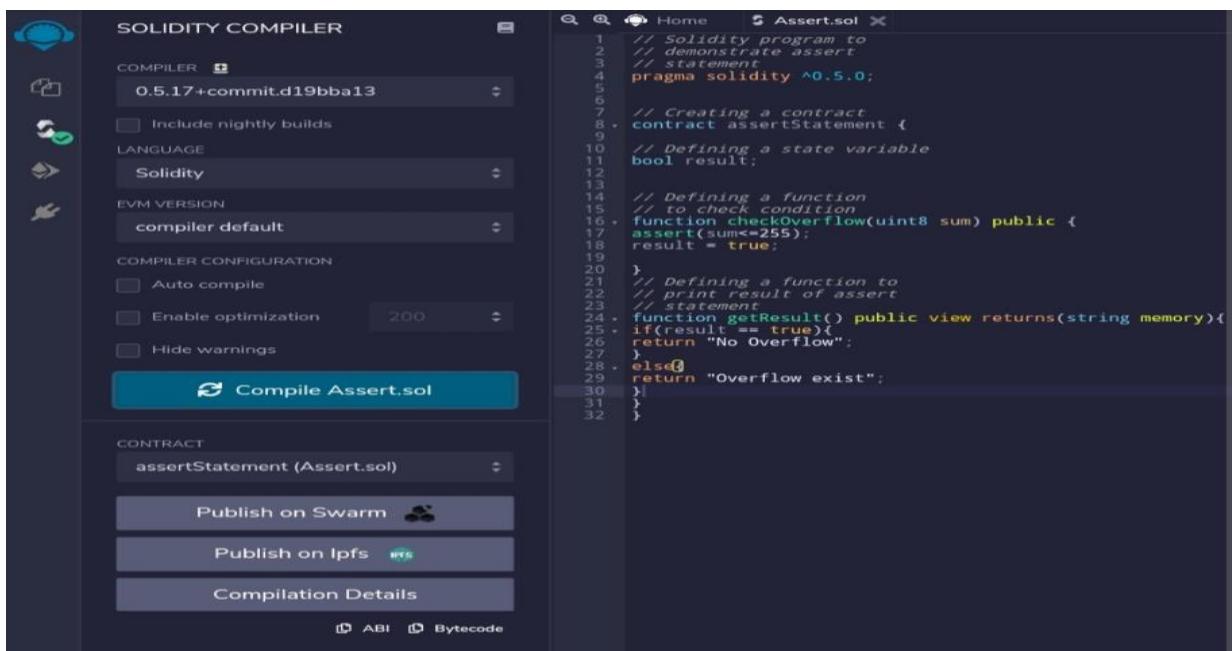
return "Overflow exist";

}

}

```

Output:



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

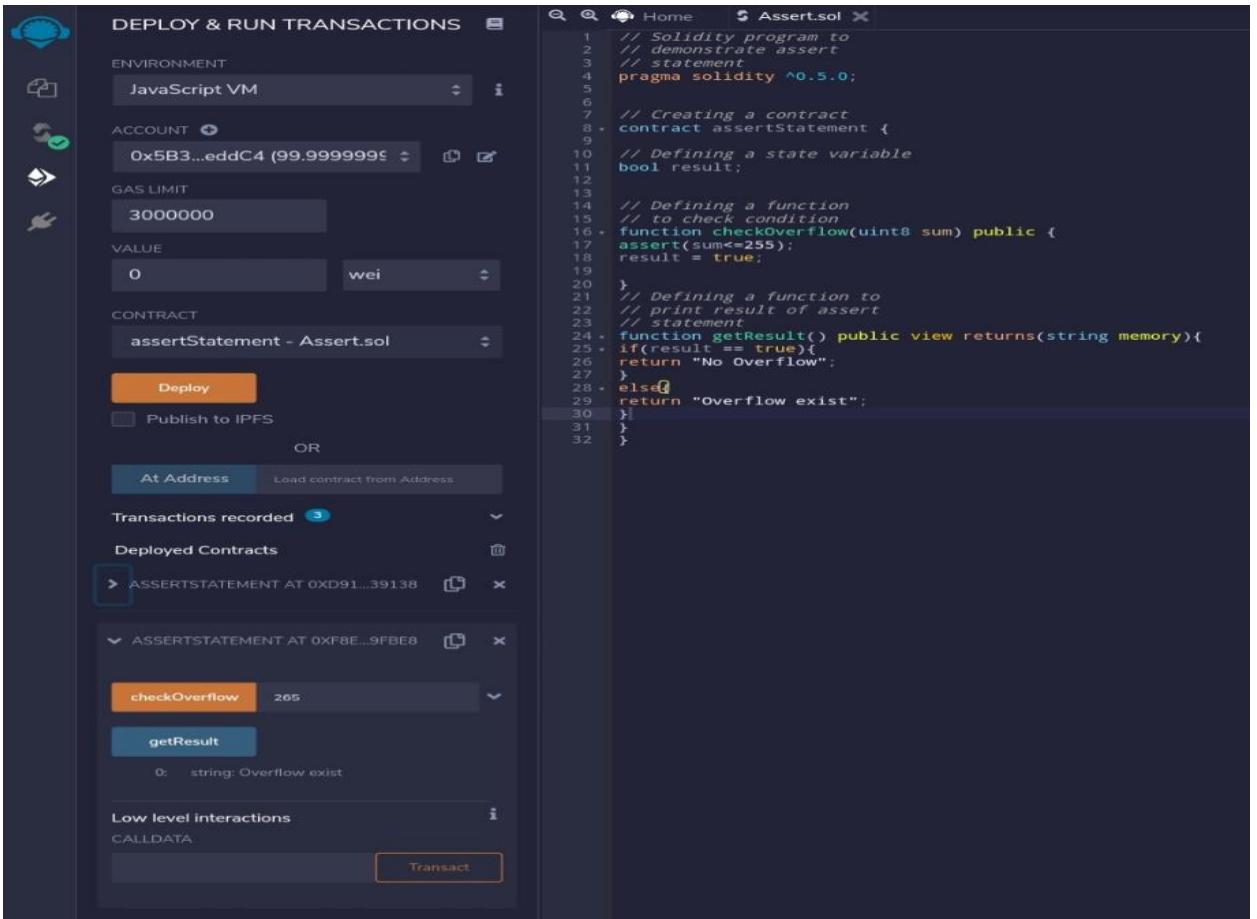
- COMPILER:** 0.5.17+commit.d19bba13
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:**
 - Auto compile
 - Enable optimization: 200
 - Hide warnings
- CONTRACT:** assertStatement (Assert.sol)
- BUTTONS:** Publish on Swarm, Publish on Ipfs, Compilation Details
- TOOLS:** ABI, Bytecode

The code editor on the right contains the Solidity code for the `Assert.sol` contract.

```

1 // Solidity program to
2 // demonstrate assert
3 // statement
4 pragma solidity ^0.5.0;
5
6
7 // Creating a contract
8 contract assertStatement {
9
10 // Defining a state variable
11 bool result;
12
13
14 // Defining a function
15 // to check condition
16 function checkOverflow(uint8 sum) public {
17     assert(sum<=255);
18     result = true;
19 }
20
21 // Defining a function to
22 // print result of assert
23 // statement
24 function getResult() public view returns(string memory){
25     if(result == true){
26         return "No Overflow";
27     }
28     else{
29         return "Overflow exist";
30     }
31 }
32

```



// 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{

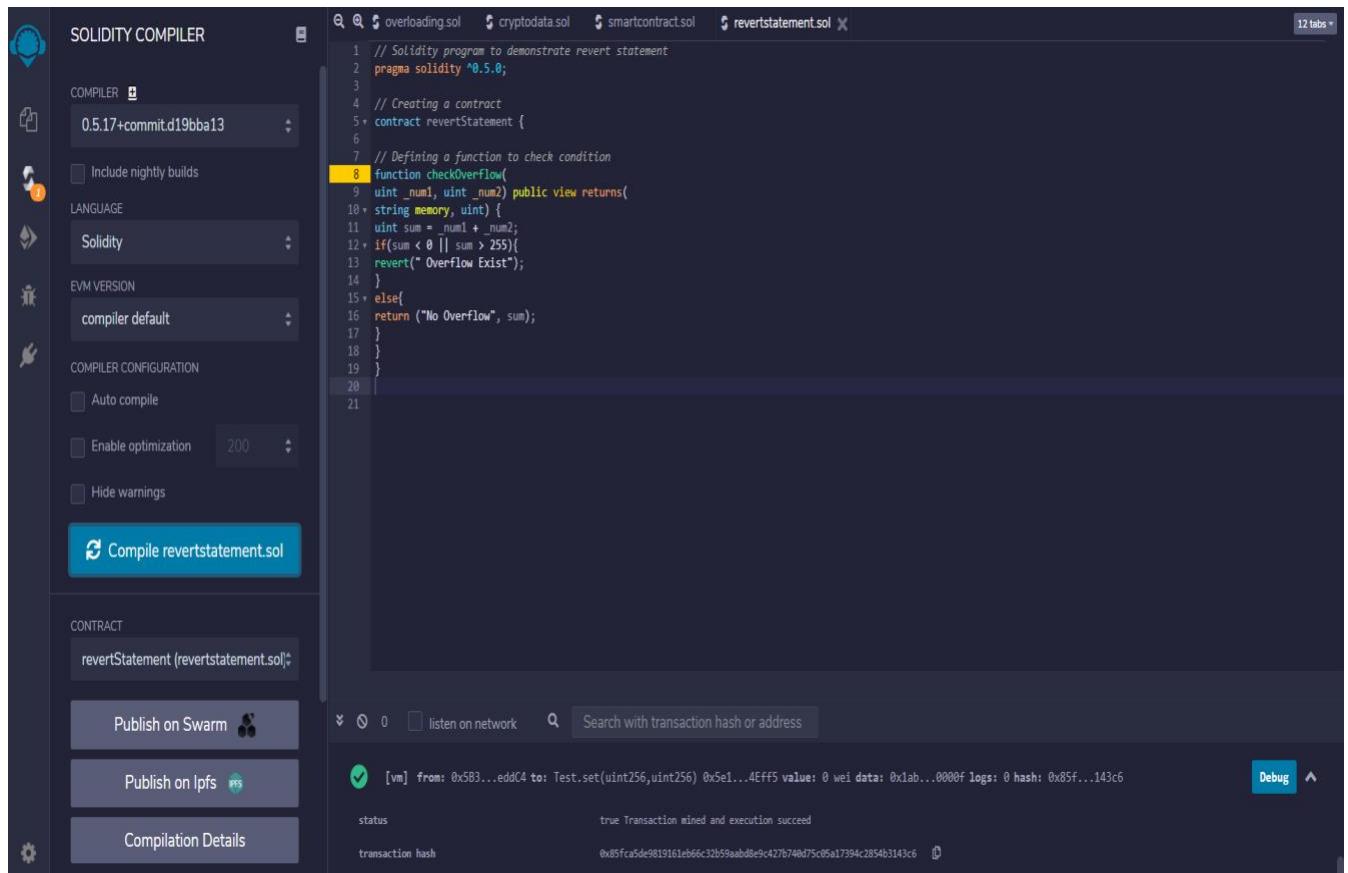
return ("No Overflow", sum);

}

}

```

Output:



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

- SOLIDITY COMPILER** tab is selected.
- COMPILER**: Version 0.5.17+commit.d19bba13
- LANGUAGE**: Solidity
- EVM VERSION**: compiler default
- COMPILE CONFIGURATION**: Auto compile, Enable optimization (200), Hide warnings
- Contracts**: revertStatement (revertstatement.sol)*
- Buttons**: Publish on Swarm, Publish on Ipfs, Compilation Details
- Logs** section shows a successful transaction with status "true Transaction mined and execution succeed" and transaction hash 0x85fca5de9819161eb66c32b59aab8e9c427b740d75c05a17394c2854b3143c6

```

1 // Solidity program to demonstrate revert statement
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5 contract revertStatement {
6
7 // Defining a function to check condition
8 function checkOverflow(
9     uint _num1, uint _num2) public view returns(
10    string memory, uint) {
11    uint sum = _num1 + _num2;
12    if(sum < 0 || sum > 255){
13        revert(" Overflow Exist");
14    }
15    else{
16        return ("No Overflow", sum);
17    }
18 }
19 }
20

```

The screenshot shows the Truffle UI interface for interacting with a Ethereum blockchain. The main area displays a Solidity smart contract named `revertstatement.sol`. The code implements a function `checkOverflow` that checks if the sum of two uint variables exceeds 255, reverting the transaction if so.

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

contract revertStatement {
    // Creating a contract
    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);
        }
    }
}
```

The interface includes sections for "DEPLOY & RUN TRANSACTIONS" and "Low level interactions". In the "Low level interactions" section, a transaction is being simulated with inputs `_num1: 52` and `_num2: 35`. A "call" button is present to execute the transaction. The results show that the transaction was successful with a status of "true Transaction mined and execution succeed" and a transaction hash of `0x@5fc5de9819161eb66c32b59aabde9-427b740d75c05a17394c285d3143c6`.