

Index

Sr. No	Topic	Page No.	Signature
1	Practical 1 Write the following programs for Blockchain in Python :	3	
	a. A simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it		
	b. A transaction class to send and receive money and test it.		
2	Practical 2 Write the following programs for Blockchain in Python :	7	
	a. Create multiple transactions and display them.		
	b. Create a blockchain, a genesis block and execute it.		
3	Practical 3 Write the following programs for Blockchain in Python :	13	
	a. Create a mining function and test it.		
	b. Add blocks to the miner and dump the blockchain.		
4	Practical 4 Implement and demonstrate the use of the following in Solidity:	20	
	a. Varaible		
	b. Operators		
	c. Loops		
	d. Decision Making		
	e. Strings		
5	Practical 5 Implement and demonstrate the use of the following in Solidity :	36	
	a. Arrays		
	b. Enums		
	c. Structs		
	d. Mappings		
	e. Coversations		
	f. Ether Units		
	g. Special Variables		

Index

6		Practical 6 Implement and demonstrate the use of the following in Solidity :	44	
	a.	Functions		
	b.	View Functions		
	c.	Pure Functions		
	d.	Fallback Functions		
	e.	Function Overloading		
	f.	.Mathematical Functions		
	g.	Cryptographic Functions		
7		Practical 7 Implement and demonstrate the use of the following in Solidity :	55	
	a.	Contracts		
	b.	Inheritance		
	c.	Constructors		
	d.	Abstract Class		
	e.	Interfaces		
8		Practical 8 Implement and demonstrate the use of the following in Solidity :	61	
	a.	Libraries		
	b.	Assembly		
	c.	Events		
	d.	Error Handling		

Practical 1

Write the following programs for Blockchain in Python:

- a) A simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it
- b) 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 two sessions in Google Colab. The top session is titled 'BLOCK CHAIN pract 1(A).ipynb'. It contains the following code in 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
```

Cell [3] shows the command to install pycryptodome:

```
[3] pip install pycryptodome
```

Output of cell [3]:

```
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
```

The bottom session is also titled 'BLOCK CHAIN pract 1(A).ipynb' and has 'All changes saved'. It contains the following code in cell [5]:

```
[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()
```

A status bar at the bottom of the interface says 'Waiting for colab.research.google.com...'.

CO BLOCK CHAIN pract 1(A).ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
    self.value = value
    self.time = datetime.datetime.now()
def to_dict(self):
    if self.sender == "Genesis":
        identity = "Genesis"
    else:
        identity = self.sender.identity
    return collections.OrderedDict([
        'sender': identity,
        'recipient': self.recipient,
        'value': self.value,
        'time' : self.time])
def sign_transaction(self):
    private_key = self.sender._private_key
    signer = PKCS1_v1_5.new(private_key)
    h = SHA.new(str(self.to_dict()).encode('utf8'))
    return binascii.hexlify(signer.sign(h)).decode('ascii')

Dinesh = Client()
Ramesh = Client()

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

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

656f0e37653d1bc3c538de01595a9c7e4a603e03740d1ed450f1ea32f568663cebff7ec9757cdd35d4f8b1a3524341528a48674981265b68a477c06d57271d6250f45d6e3946a4f8166fff50251956fa

Waiting for colab.research.google.com...

Practical 2

Write the following programs for Blockchain in Python:

- a) Create multiple transactions and display them.
- b) 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
)
```

```

[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
-----
-----  

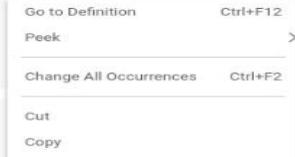
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a3b073014e3d9a1e335365f
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5
-----
value: 4.0
-----
time: 2021-05-05 07:38:17.045503
-----
-----  

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:

- a) Create a mining function and test it.
- b) 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
|██████████| 1.9MB 5.6MB/s
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.10.1

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

[ ] import binascii
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer = PKCS1_v1_5.new(self._private_key)

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

class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()
    def to_dict(self):
        if self.sender == "Genesis":
            identity = "Genesis"
        else:
            identity = self.sender.identity
        return collections.OrderedDict({
            'sender': identity,
            'recipient': self.recipient,
            'value': self.value,
            'time' : self.time})
    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')

[ ]

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

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

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

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: 30819f300d06092a864886f70d010101050003818d00308189028181008e40f44f9cef42cac49
    -----
    value: 15.0
    -----
    time: 2021-04-08 06:10:52.172517
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4713
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 6.0
    -----
    time: 2021-04-08 06:11:40.567785
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d00308189028181008e40f44f9cef42cac492d1
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e
    -----
    value: 2.0
    -----
    time: 2021-04-08 06:11:40.569278
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86384
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d00308189028181008e40f44f9cef42cac49
    -----
    value: 4.0
    -----
    time: 2021-04-08 06:11:40.570658
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 7.0
    -----
    time: 2021-04-08 06:11:40.572173
    -----
    -----
    sender: 30819f300d06092a864886f70d010101050003818d00308189028181008e40f44f9cef42cac492d1
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 3.0
    -----
    time: 2021-04-08 06:11:40.574026

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

[ ]     last_block_hash = ""

[ ]     Dinesh = Client()

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

[ ]     block0 = Block()

[ ]     block0.previous_block_hash = None
[ ]    Nonce = None

[ ]     block0.verified_transactions.append (t0)

```

+ Code + Text | 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

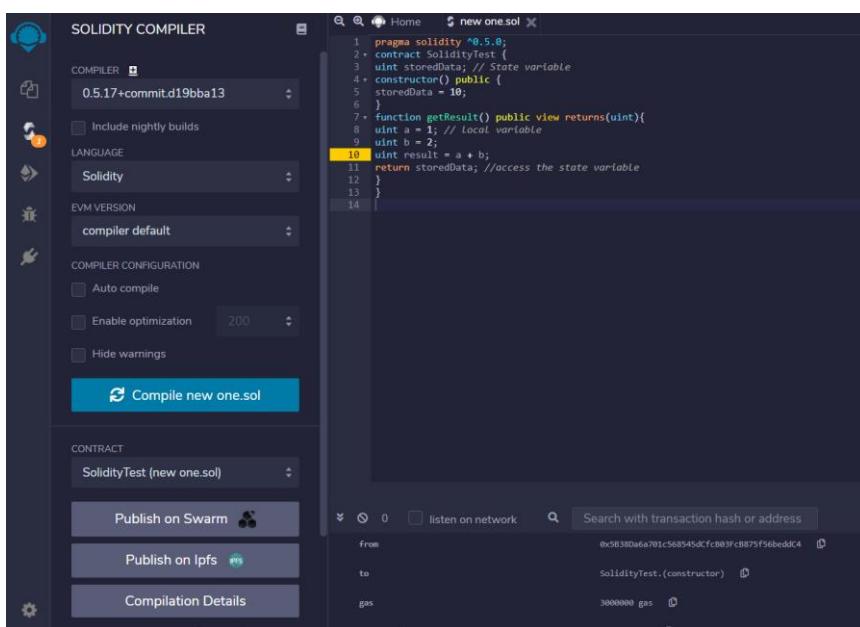
Implement and demonstrate the use of the following in Solidity:

- a) Variable
- b) Operators
- c) Loops
- d) Decision Making
- e) Strings

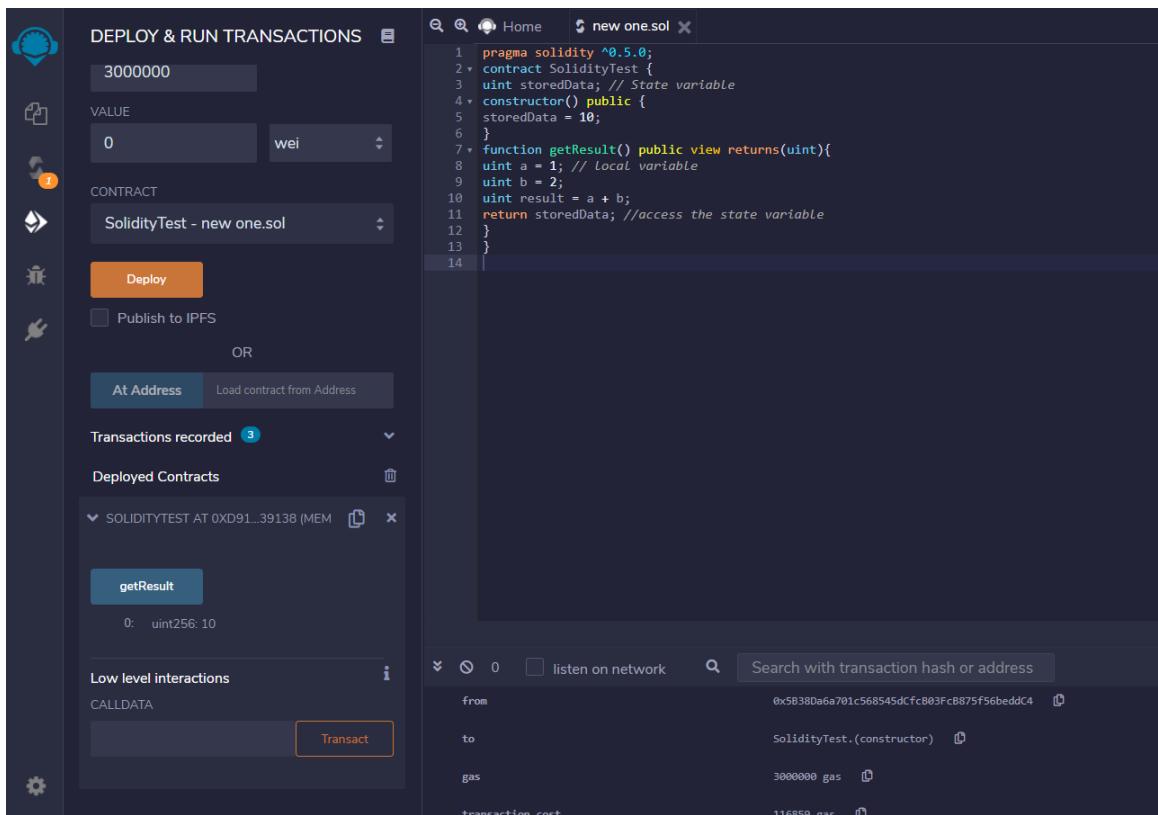
a.Variable

```
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:



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), and EVM version (compiler default). Under 'COMPILER CONFIGURATION', there are checkboxes for 'Auto compile', 'Enable optimization' (set to 200), and 'Hide warnings'. A blue button at the bottom left says 'Compile new one.sol'. In the center, the Solidity code is displayed. The line 'uint result = a + b;' is highlighted in yellow. To the right, the compiled Solidity code is shown, along with the transaction details for deploying the contract: from address 0x56380a6a701c568545d7fc803fcb875f56bedd04, to address SolidityTest.(constructor), and gas limit 3000000.



// Solidity program to demonstrate state variables

```
pragma solidity ^0.5.0;
// Creating a contract
contract Solidity_var_Test {
// Declaring a state variable
uint8 public state_var;
// Defining a constructor
constructor() public {
state_var = 16;
}
}
```

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:
 - Compile button: **Compile new one.sol**- CONTRACT** section:
 - Solidity_var_Test (new one.sol)
 - Publish on Swarm:
 - Publish on Ipfs:
 - Compilation Details
- Code Editor** (new one.sol):

```
1 // Solidity program to demonstrate state variables
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5 contract Solidity_var_Test {
6
7 // Declaring a state variable
8 uint8 public state_var;
9
10 // Defining a constructor
11 constructor() public {
12 state_var = 16;
13 }
14 }
```
- Deployment Details**:
 - from: 0x5B38Da6a701c568545dCfcB03FcB875
 - to: SolidityTest.(constructor)
 - gas: 3000000 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 (unchecked)
 - Enable optimization (unchecked)
 - Hide warnings (unchecked)
- CONTRACT:** Solidity_var_Test (Variable.sol)
- Buttons:** Compile Variable.sol, Publish on Swarm, Publish on Ipfs, Compilation Details, ABI, Bytecode.

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 (99.99999999999999)
- 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
 - state_var:** 0: uint8: 16
- Low level interactions:** CALLDATA, Transact

The code editor on the right is identical to the one in the Solidity Compiler interface.

```
// 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:

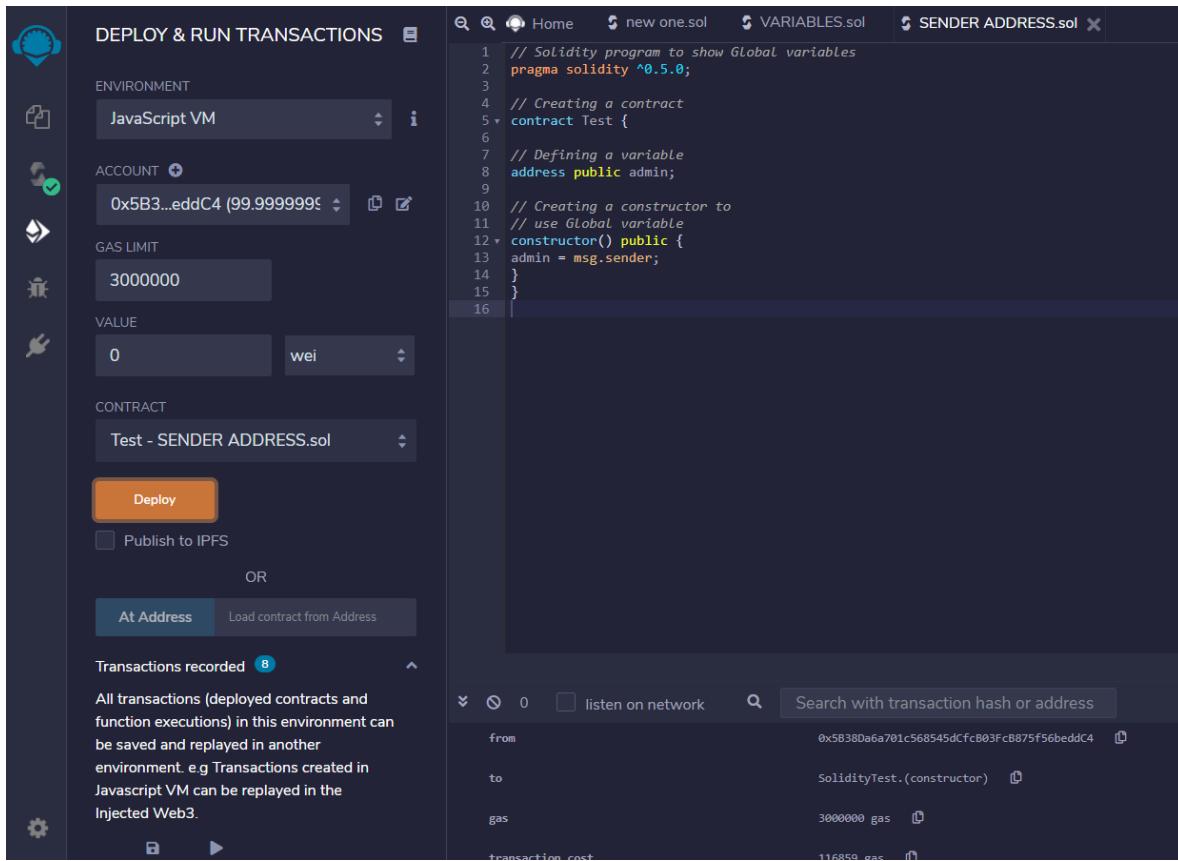
The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons. The main area has tabs for 'SOLIDITY COMPILER' and 'VARIABLES.sol'. The 'SOLIDITY COMPILER' tab is active, displaying compiler settings like version (0.5.17+commit.d19bba13), language (Solidity), and EVM version (compiler default). It also includes compiler configuration options for auto-compile, optimization level (200), and warning visibility. A prominent blue button at the bottom says 'Compile SENDER ADDRESS.sol'. The right side shows the Solidity code in a code editor. Below the code editor is a transaction details section where a transaction is being prepared, showing fields for 'from', 'to', and 'gas'. The 'to' field is set to 'SolidityTest.(c...'.

```
// 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;
}
}
```



b. 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. On the left, there's a sidebar with compiler settings like version (0.5.17+commit.d19bba13), language (Solidity), and EVM version (compiler default). Below that are compiler configuration options: Auto compile, Enable optimization (set to 200), and Hide warnings. A prominent blue button at the bottom of the sidebar says "Compile arithmetic.sol". The main area displays the Solidity code for "arithmeticTest.sol". The code initializes variables a and b, calculates sum, difference, product, quotient, modulus, and decrements/increments them. At the bottom of the code editor, there are tabs for Home, new one.sol, VARIABLES.sol, SENDER ADDRESS.sol, and arithmetic.sol. The bottom section shows the "CONTRACT" tab selected, displaying "SolidityTest (arithmetic.sol)". It includes buttons for "Publish on Swarm" and "Publish on Ipfs", and a "Compilation Details" section. The "Compilation Details" section shows the transaction hash (0x5830de6a701c568545cfcb003fc8875f56bedd4), the contract address (SolidityTest.(constructor)), and gas limit (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 = ++a;
30
31

```

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.99999999999999)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: SolidityTest - arithmetic.sol

Deploy

Publish to IPFS

OR

At Address **Load contract from Address**

Transactions recorded 9

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.

```

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 = ++a;
30
31

```

From: 0x5B3...eddC4

To: SolidityTest.(constructor)

Gas: 3000000 gas

DEPLOY & RUN TRANSACTIONS

SOLIDITYTEST AT 0xD2A...FD005 (MEN)

a

b

dec

diff

div

inc

mod

mul

sum

Low level interactions

CALDATA

Transact

```

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 = ++a;
30
31

```

From: 0x5B3...eddC4

To: SolidityTest.(constructor)

Gas: 3000000 gas

transaction cost: 116050 gas

C.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){
```

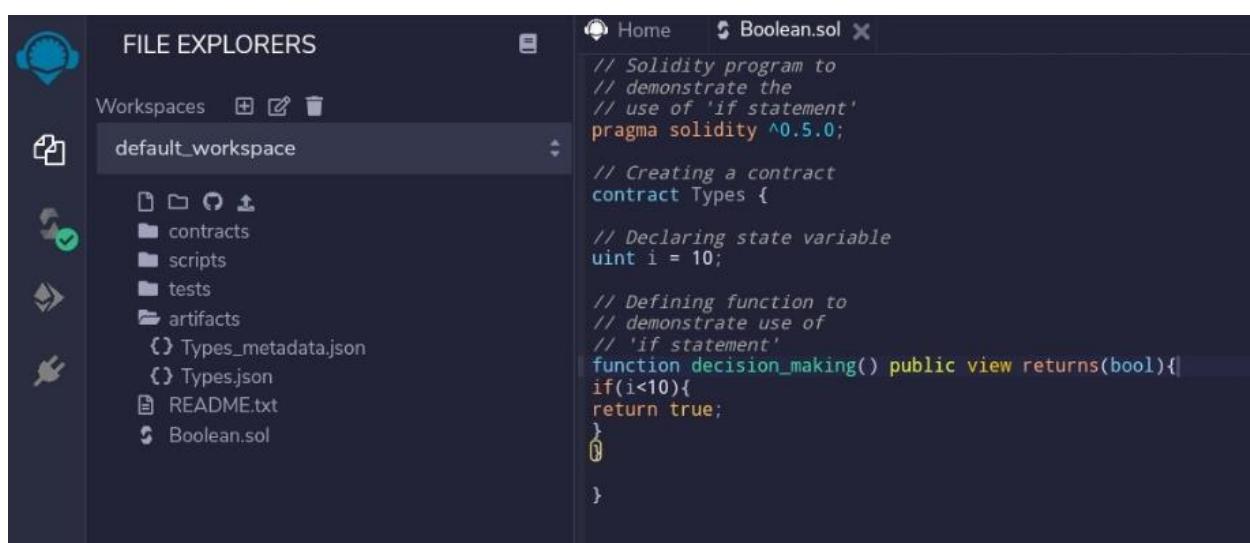
```
return true;
```

```
}
```

```
}
```

```
}
```

Output:



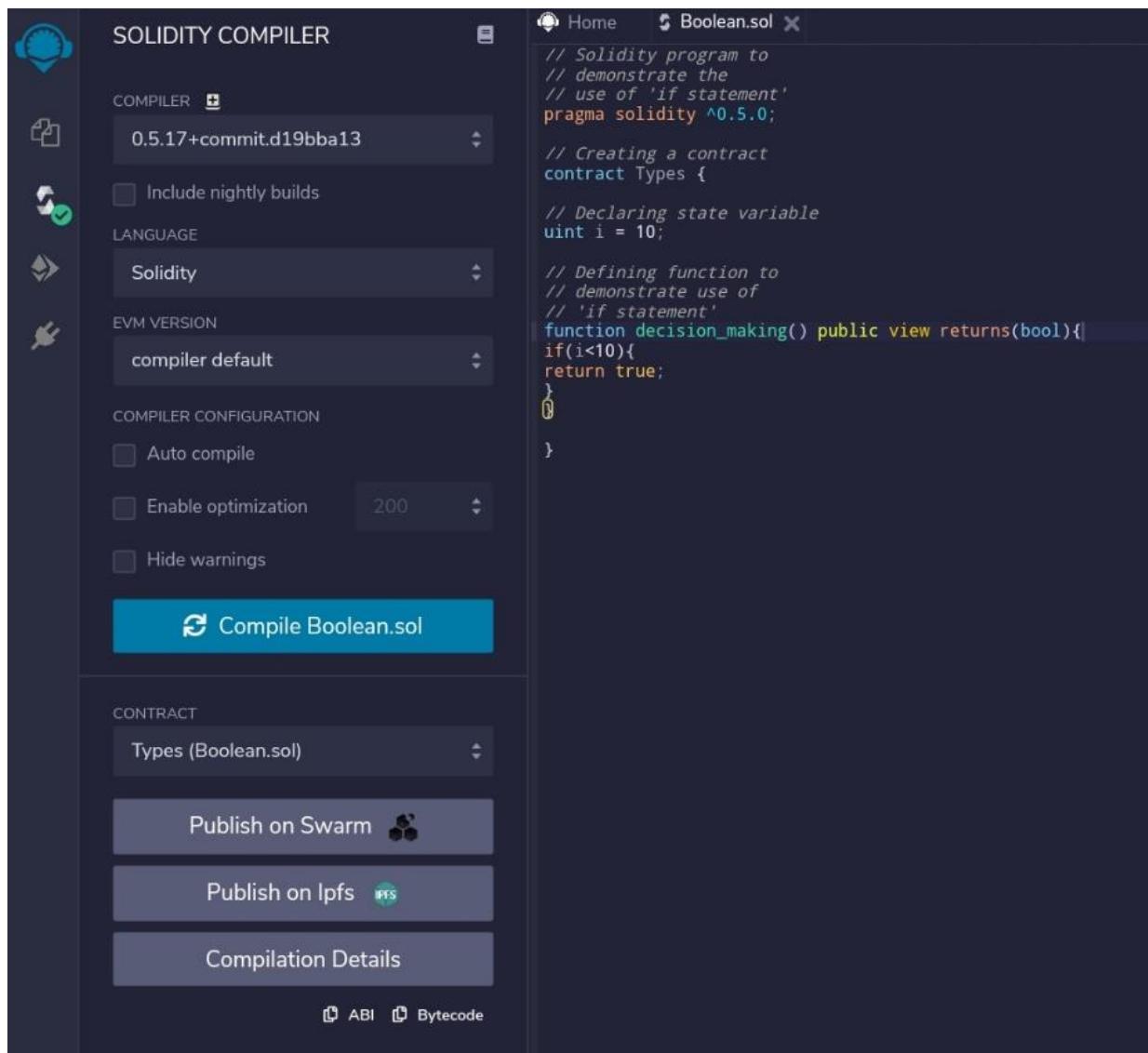
The screenshot shows the Visual Studio Code interface. On the left, the 'FILE EXPLORERS' sidebar displays a 'default_workspace' folder containing 'contracts', 'scripts', 'tests', 'artifacts', 'Types_metadata.json', 'Types.json', 'README.txt', and 'Boolean.sol'. The 'Boolean.sol' file is currently open in the main code editor area. The code editor shows the following Solidity code:

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

// Creating a contract
contract Types {

// Declaring state variable
uint i = 10;

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



The image shows the Solidity Compiler interface. On the left, there's a sidebar with various icons: a gear, a file, a checkmark, a double arrow, and a hand. The main area has a header "SOLIDITY COMPILER". Below it, under "COMPILER", is a dropdown set to "0.5.17+commit.d19bba13". There are checkboxes for "Include nightly builds", "Auto compile", "Enable optimization" (set to 200), and "Hide warnings". A large blue button at the bottom says "Compile Boolean.sol". Under "CONTRACT", there's a dropdown set to "Types (Boolean.sol)". Below it 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".

Home Boolean.sol

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

// Creating a contract
contract Types {

    // Declaring state variable
    uint i = 10;

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

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, the "DEPLOY & RUN TRANSACTIONS" panel includes fields for Environment (JavaScript VM), Account (0x5B3...eddC4), Gas Limit (3000000), Value (0 wei), and Contract (Types - Boolean.sol). A prominent orange "Deploy" button is at the bottom. Below it is a checkbox for "Publish to IPFS". The right side displays the "Boolean.sol" file content:

```
// 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;
        }
        return false;
    }
}
```

The "Transactions recorded" section shows one entry for "TYPES AT 0xD91...39138 (MEMORY)". The "Low level interactions" section includes a "Transact" button.

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

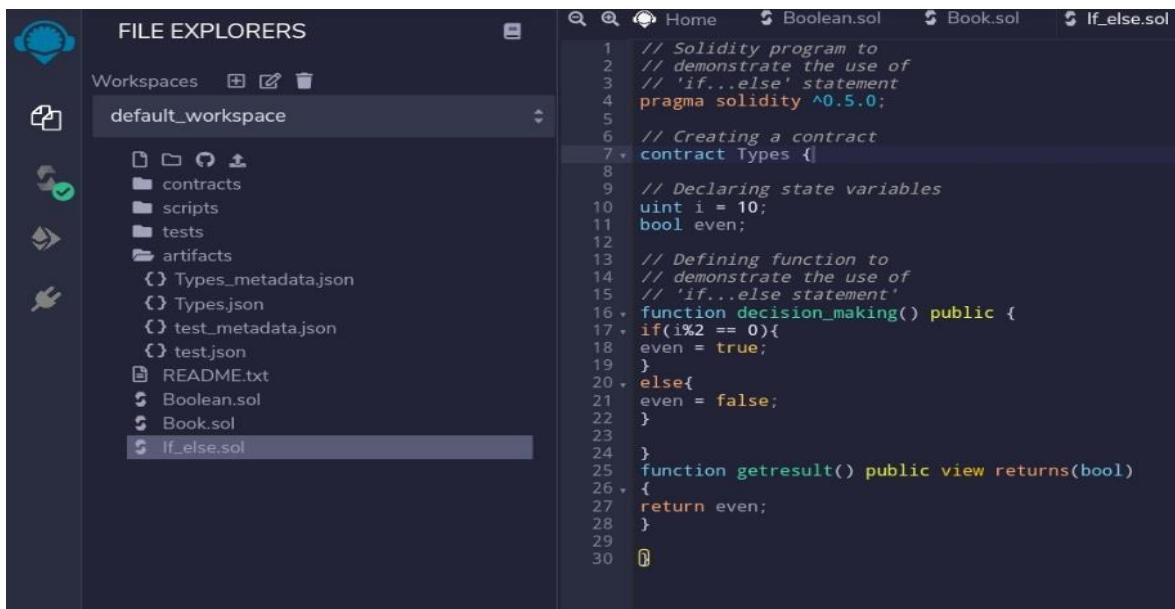
// Creating a contract
contract Types {

// 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's file explorer. On the left, there are icons for workspaces, contracts, scripts, tests, and artifacts. The 'default_workspace' is selected. Inside, there are folders for 'contracts', 'scripts', 'tests', and 'artifacts'. Under 'artifacts', files like 'Types_metadata.json', 'Types.json', 'test_metadata.json', 'test.json', 'README.txt', 'Boolean.sol', 'Book.sol', and 'If_else.sol' are listed. 'If_else.sol' is currently selected and highlighted.

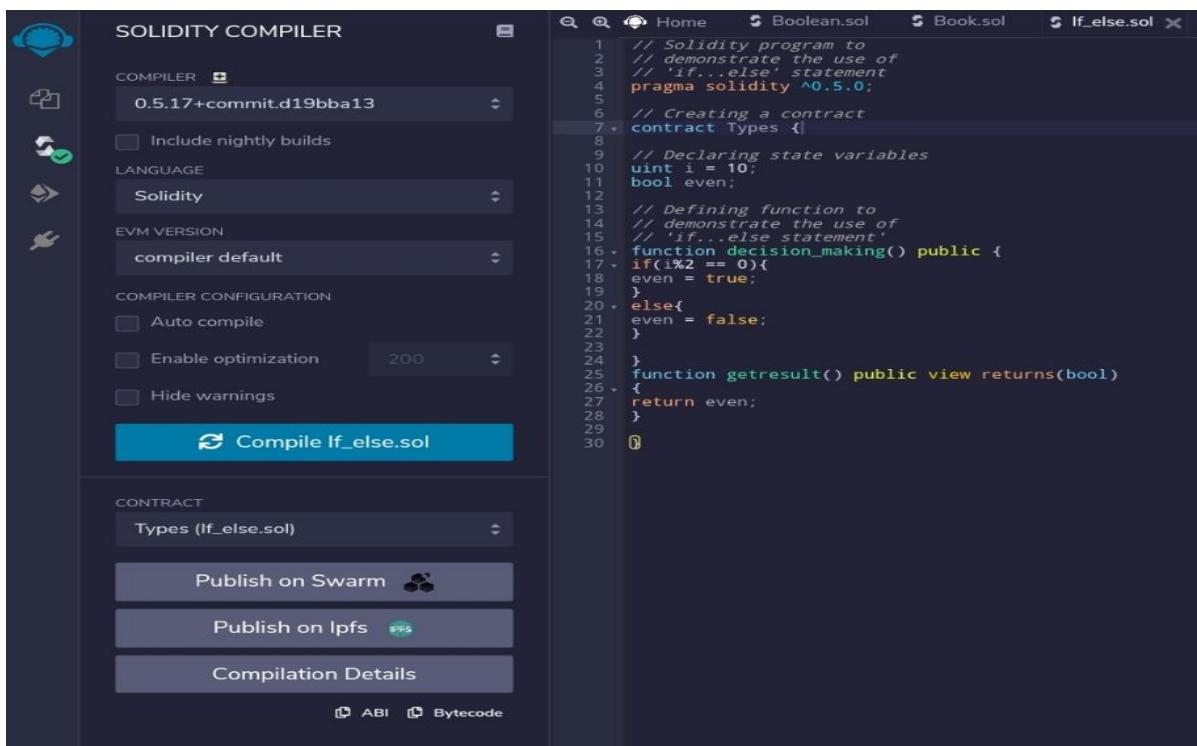
```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

// 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, it has 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 at the bottom left says 'Compile If_else.sol'. Below this, under 'CONTRACT', it shows 'Types (If_else.sol)' and three buttons: 'Publish on Swarm' (with a Swarm icon), 'Publish on Ipfs' (with an IPFS icon), and 'Compilation Details'. At the very 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. On the left, the 'DEPLOY & RUN TRANSACTIONS' tab is active. It includes fields for ENVIRONMENT (JavaScript VM), ACCOUNT (0x5B3...eddC4), GAS LIMIT (3000000), and VALUE (0 wei). Under CONTRACT, 'Types - If_else.sol' is selected. A large orange 'Deploy' button is prominent. Below it, there's a checkbox for 'Publish to IPFS'. The right side of the interface is a code editor displaying Solidity code:

```
// 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 compiler configuration (Auto compile). It also has checkboxes for optimization (Enable optimization, 200) and hiding warnings. A large blue button at the bottom says 'Compile string.sol'. Below this is the 'CONTRACT' section, which lists 'Test (string.sol)' and provides options to 'Publish on Swarm' or 'Publish on IPFS'. At the bottom of the sidebar is a 'Compilation Details' section. The main area displays the Solidity code for 'string.sol':

```
1 // Solidity program to demonstrate
2 // how to create a contract
3 pragma solidity ^0.4.23;
4
5 // Creating a contract
6+ contract Test {
7
8 // Declaring variable
9 string str;
10
11 // Defining a constructor
12+ constructor(string str_in){
13 str = str_in;
14 }
15
16 // Defining a function to
17 // return value of variable 'str'
18+ function str_out() public view returns(string memory){
19 return str;
20 }
21 }
22
```

The transaction details at the bottom show a transaction from 0x58380a6a701c568545dCfcB803fC875f56beddK4 to SolidityTest.(constructor) with 3000000 gas.

The screenshot shows the 'DEPLOY & RUN TRANSACTIONS' interface. On the left, the 'TEST AT 0X9D7...D5E99 (MEMORY)' section shows a 'state_var' interaction with a 'CALLDATA' input field containing '0'. A note says 'The calldata should be a valid hexadecimal value.' The 'TEST AT 0XD2A...FD005 (MEMORY)' section shows a 'str_out' interaction. The main area displays the same Solidity code as the previous screenshot. The transaction details at the bottom are identical to the first screenshot.

Practical 5

Implement and demonstrate the use of the following in Solidity:

- a) Arrays
- b) Enums
- c) Structs
- d) Mappings
- e) Conversations
- f) Ether Units
- g) Special Variables

(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' sidebar lists workspaces and files. The 'default_workspace' workspace is selected, showing files like README.txt, Variable.sol, Sender Address.sol, String.sol, and Array.sol. 'Array.sol' is currently selected. The main area is a code editor with the following Solidity code:

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

The screenshot shows the Solidity Compiler interface. On the left, various compiler settings are listed: COMPILER (version 0.5.17+commit.d19bba13), LANGUAGE (Solidity), EVM VERSION (compiler default), and COMPILER CONFIGURATION (Auto compile, Enable optimization set to 200). Below these are checkboxes for Hide warnings and Hide errors. A large blue button labeled 'Compile Array.sol' is prominent. The right side is a code editor tab for 'Array.sol' with the same Solidity code as the previous screenshot.

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with various icons for network selection, account management, gas limit, value, and contract selection. The main area has tabs for 'DEPLOY & RUN TRANSACTIONS' and '5 tabs'. The current tab is 'Array.sol'.

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.9999999)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: Types - Array.sol

Deploy

Publish to IPFS

OR

At Address: Load contract from Address

Transactions recorded: 3

Deployed Contracts: TYPES AT 0xD91...39138 (MEMORY)

array_example

getresult

0: int256(5): 50,-63,77,-28,90
1: uint256[6]: 10,20,30,40,50,60

Low level interactions:

CALldata

Transact

Array.sol

```
// 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;

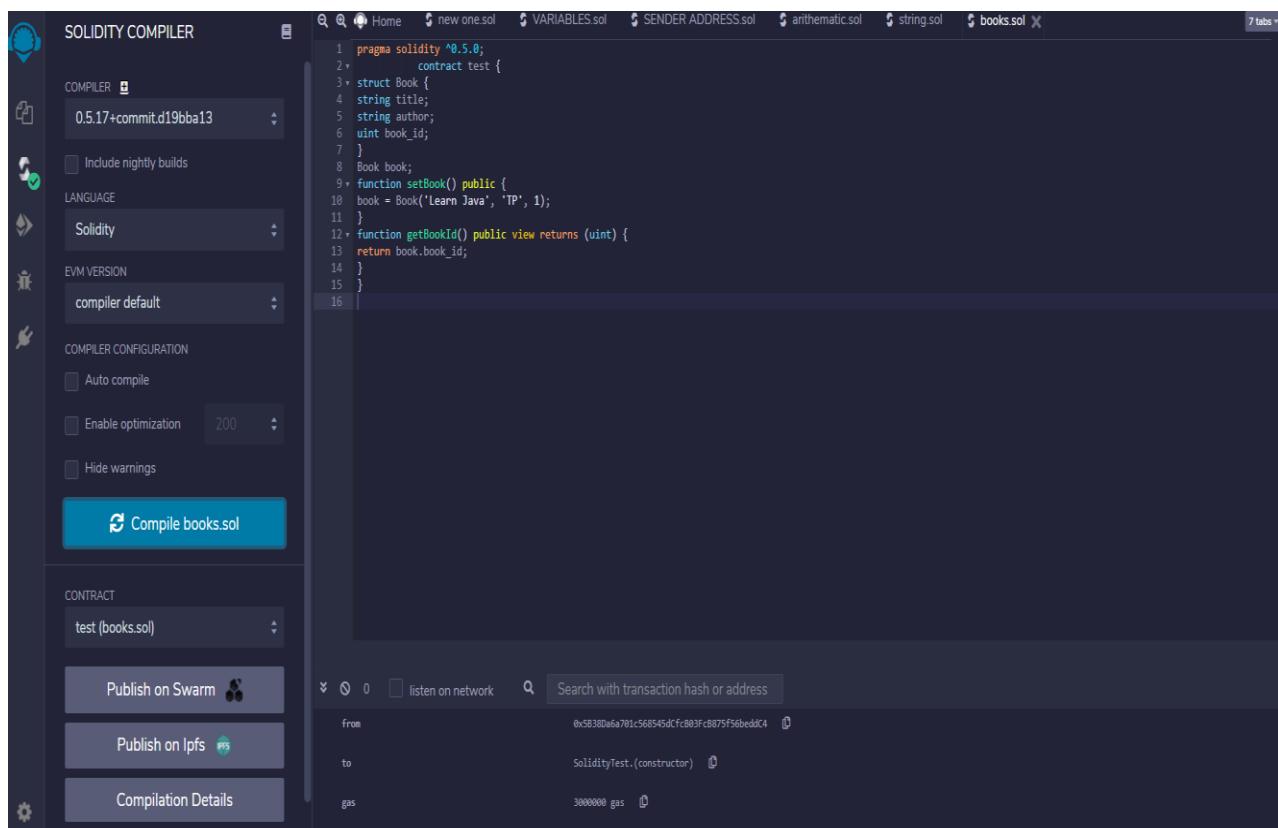
    // Defining function to add
    // values to an array
    function array_example() public returns (int[5] memory, uint[6] memory){
        data = [int(50), -63, 77, -28, 90];
        data1 = [uint(10), 20, 30, 40, 50, 60];
    }

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

c.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 Truffle IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Delete. Below that is a section for "Low level interactions" with a "CALLDATA" tab and a "Transact" button. The main area contains several tabs for different test environments:

- TEST AT 0x9D7...B5E99 (MEMORY)**: Shows a Solidity contract named "test" with a struct "Book" and two functions: "setBook()" and "getBookId()".
- SOLIDITYTEST AT 0xD2A...FD005 (MEMORY)**: Shows the same contract "test" with the same functions.
- TEST AT 0XDDA...5492D (MEMORY)**: Shows the contract "test" with the same functions. It includes a "str_out" button and a "Low level interactions" section with a "CALLDATA" tab and a "Transact" button.
- TEST AT 0X0FC...9A836 (MEMORY)**: Shows the contract "test" with the same functions. It includes a "setBook" button and a "getBookId" button. The "getBookId" button has a dropdown menu showing "0: uint256 1".

At the bottom, there's a search bar with the placeholder "Search with transaction hash or address" and a "transaction cost" section showing "0 gas" and "116859 gas".

Mappings

```

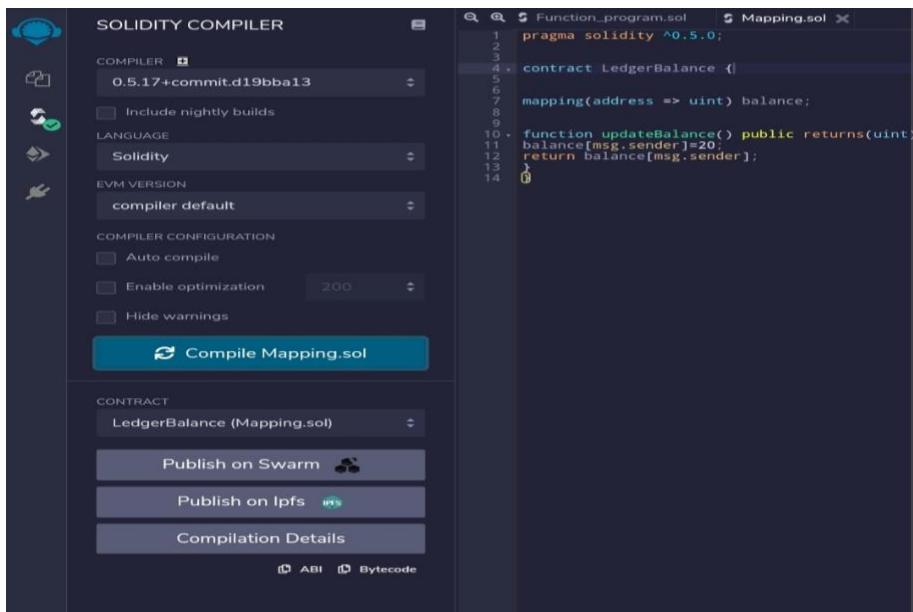
pragma solidity ^0.5.0;

contract LedgerBalance {
    mapping(address => uint) balance;

    function updateBalance() public returns(uint) {
        balance[msg.sender]=20;
        return balance[msg.sender];
    }
}

```

Output:



The Solidity Compiler interface shows the following configuration:

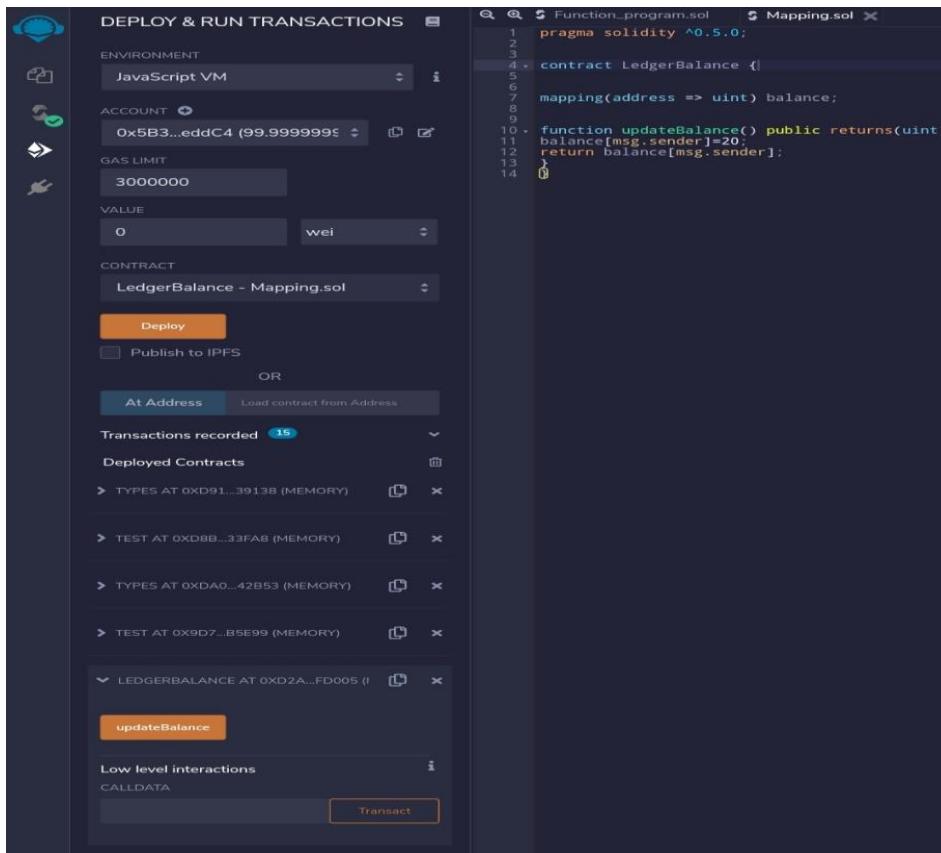
- Compiler: 0.5.17+commit.d19bba13
- Include nightly builds: unchecked
- Language: Solidity
- EVM Version: compiler default
- Compiler Configuration:
 - Auto compile: unchecked
 - Enable optimization: checked (value 200)
 - Hide warnings: unchecked

Contract: LedgerBalance (Mapping.sol) is selected.

Buttons:

- Compile Mapping.sol
- Publish on Swarm
- Publish on Ipfs
- Compilation Details

ABI and Bytecode links are also present.



The Deploy & Run Transactions interface shows the following configuration:

- Environment: JavaScript VM
- Account: 0x5B3...eddC4 (99.9999995)
- GAS LIMIT: 3000000
- Value: 0 wei

Contract: LedgerBalance - Mapping.sol is selected.

Buttons:

- Deploy
- Publish to IPFS

OR

At Address: Load contract from Address

Transactions recorded: 35

Deployed Contracts:

- TEST AT 0XD8B...33FA8 (MEMORY)
- TEST AT 0X9D7...B5E99 (MEMORY)
- LEDGERBALANCE AT 0XD2A...FD005 (MEMORY)

Low level interactions:

- CALldata

Buttons:

- updateBalance
- Transact

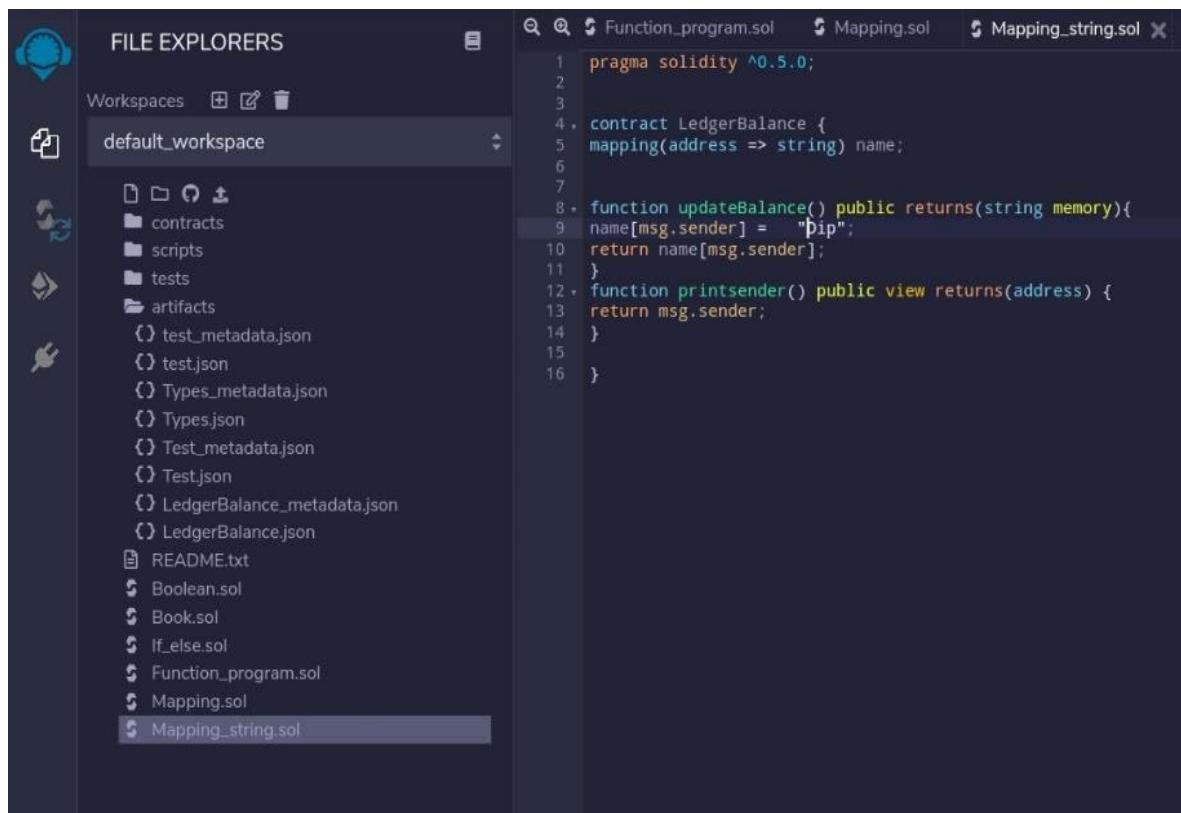
```

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 Visual Studio Code interface. On the left, the 'FILE EXPLORERS' sidebar displays a 'default_workspace' folder containing various files and folders such as contracts, scripts, tests, artifacts, and metadata files. In the center, the code editor displays the Solidity smart contract code provided above. The tabs at the top of the code editor show 'Function_program.sol', 'Mapping.sol', and 'Mapping_string.sol'. The 'Function_program.sol' tab is active.

```

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 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: 200
 - Hide warnings
- Contract:** LedgerBalance (Mapping_string.sol)
- Buttons:** Publish on Swarm, Publish on Ipfs, Compilation Details, ABI, Bytecode.

The interface displays the following details for the LedgerBalance contract:

- TEST AT 0XD8B...33FAB (MEMORY)**
- TYPES AT 0XDAO...42B53 (MEMORY)**
- TEST AT 0X9D7...B5E99 (MEMORY)**
- LEDGERBALANCE AT 0XD2A...FD005 (MEMORY)**
- LEDGERBALANCE AT 0X332...D4B6D (MEMORY)**

Functions:

- updateBalance**
- printsender**

Low level interactions:

0: address: 0x5B38Da6a701c568545dCfcB
03FcB875f56beddC4

CALLDATA:

Transact:

Practical 6

Implement and demonstrate the use of the following in Solidity:

- a) Functions
- b) View Functions
- c) Pure Functions
- d) Fallback Functions
- e) Function Overloading
- f) Mathematical Functions
- g) Cryptographic Functions

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

SOLIDITY COMPILER:

- Compiler:** 0.5.17+commit.d19bba13
- Language:** Solidity
- EVM Version:** compiler default
- Compiler Configuration:** Auto compile, Enable optimization (set to 200), Hide warnings
- Contract:** SolidityTest (Function test.sol)
- Buttons:** Publish on Swarm, Publish on Ipfs, Compilation Details
- ABI and Bytecode:** Links for ABI and Bytecode generation.

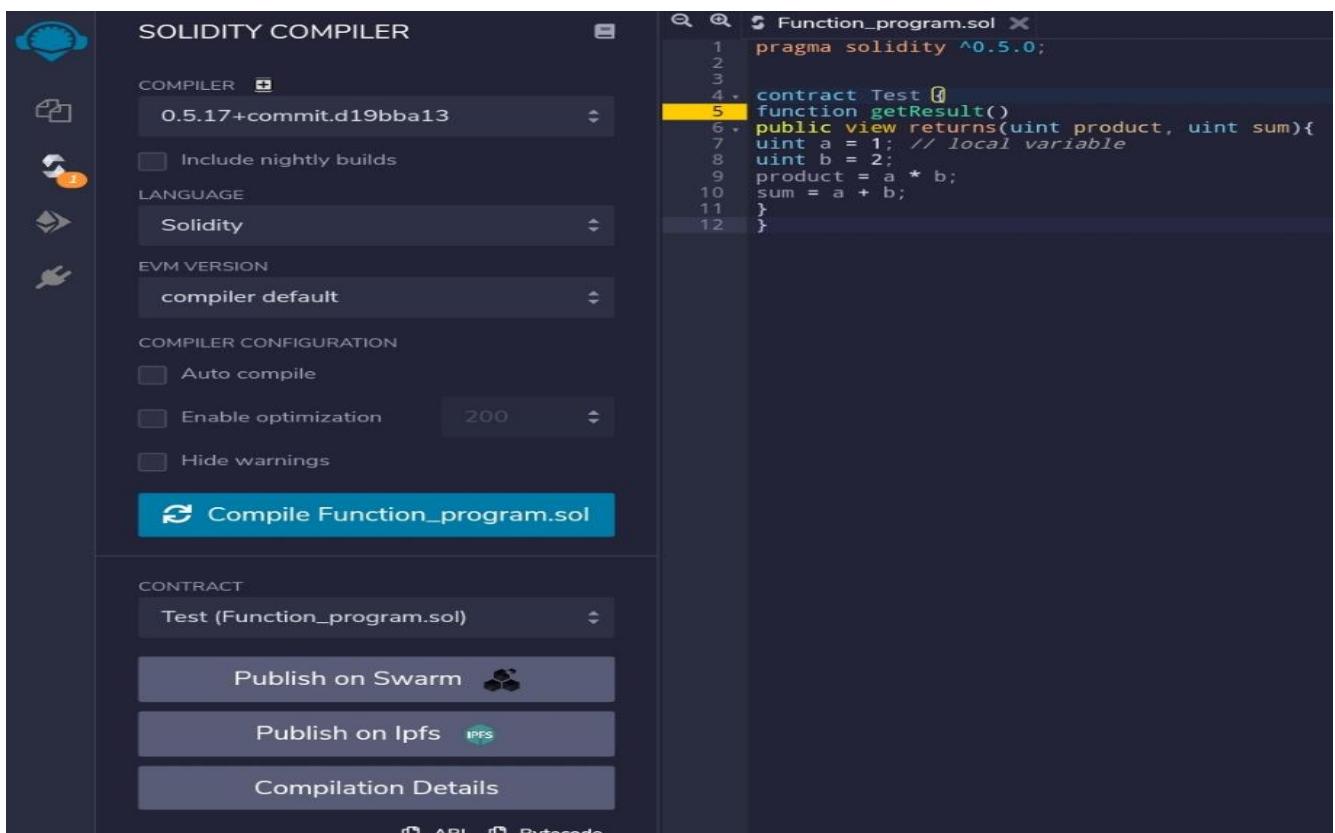
DEPLOY & RUN TRANSACTIONS:

- Environment:** JavaScript VM
- Account:** 0x5B3...eddC4 (99.99999999999999)
- Gas Limit:** 3000000
- Value:** 0 wei
- Contract:** SolidityTest - Function test.sol
- Deploy Button:** Orange button to deploy the contract.
- Publish to IPFS:** Checkbox for publishing to IPFS.
- OR:** Options to Deploy at Address or Load contract from Address.
- Transactions recorded:** 2 transactions recorded.
- Deployed Contracts:**
 - SOLIDITYTEST AT 0xD91...39138 (MEM)
 - SOLIDITYTEST AT 0xDBB...33FA8 (MEM)
 - testpgmresult:** Returns uint256, 3000
- Low level interactions:** CALldata field with a Transaction button.

B.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 various icons. The main area has tabs for 'SOLIDITY COMPILER' and 'CONTRACT'. Under 'SOLIDITY COMPILER', the compiler version is set to '0.5.17+commit.d19bba13'. The code editor on the right contains the Solidity code provided above. Below the code editor, there are sections for 'COMPILER CONFIGURATION' (Auto compile, Enable optimization, Hide warnings) and 'CONTRACT' (Test (Function_program.sol), Publish on Swarm, Publish on Ipfs, Compilation Details). At the bottom, there are links for 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 and running transactions. On the left, there's a sidebar with various icons. The main area has several input fields: ENVIRONMENT (JavaScript VM), ACCOUNT (0x5B3...eddC4), GAS LIMIT (3000000), and VALUE (0 wei). Below these are sections for CONTRACT (Test - Function_program.sol) and a Deploy button. There's also a checkbox for Publish to IPFS. Underneath, there's an OR section with tabs for At Address and Load contract from Address. The Deployed Contracts section lists several contracts with their addresses and memory locations: TYPES AT 0XD91...39138 (MEMORY), TEST AT 0XD8B...33FA8 (MEMORY), TYPES AT 0XDA0...42B53 (MEMORY), and TEST AT 0X9D7...B5E99 (MEMORY). The TEST AT 0X9D7...B5E99 entry is expanded, showing a getResult button and two return values: 0: uint256: product 2 and 1: uint256: sum 3. On the right, a code editor window displays the Solidity source code for the Test contract:

```
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;
    }
}
```

C.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

```

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(); }
}

```

CONTRACT

C (calling function external.sol)

Publish on Swarm

Publish on lpfss

Compilation Details

DEPLOY & RUN TRANSACTIONS

TEST AT 0x0FC_9A836 (MEMORY)

setBook

getBookId

0: uint256:1

Low level interactions

CALLDATA

Transact

C AT 0xAE0_96B8B (MEMORY)

updateData

25

transact

getData

0: uint256:25

info

0: uint256:10

from 0x58380a6a701c568545dCfcB03Fc887f56beddC4

to SolidityTest.(constructor)

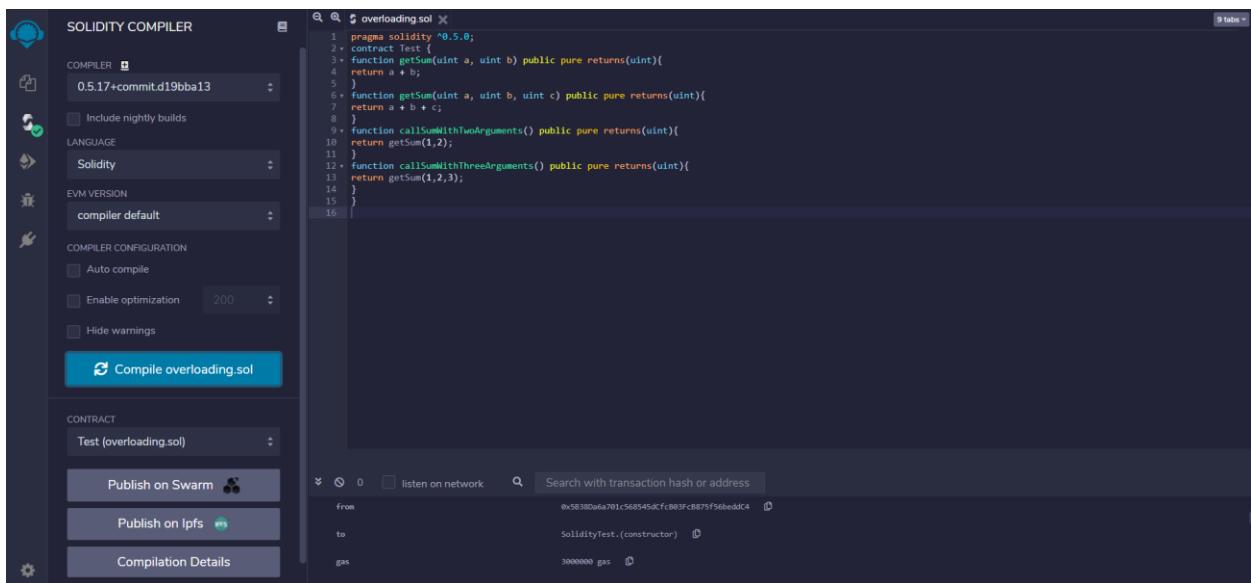
gas 3000000 gas

Search with transaction hash or address

D.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 Solidity Compiler interface with the following configuration:

- COMPILER:** 0.5.17+commit.d19bba13
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILE CONFIGURATION:** Auto compile, Enable optimization (set to 200), Hide warnings
- Contract:** Test (overloading.sol)
- Buttons:** Compile overloading.sol, Publish on Swarm, Publish on IPFS, Compilation Details
- Network Tab:** Shows a transaction hash: 0x5380a6a701c568545dcfc803fc887f56bedd04
- Gas Input:** gas = 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);
    }
}
```

```

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);
    }
}

```

E.Mathematical Functions

```

pragma solidity ^0.5.0;

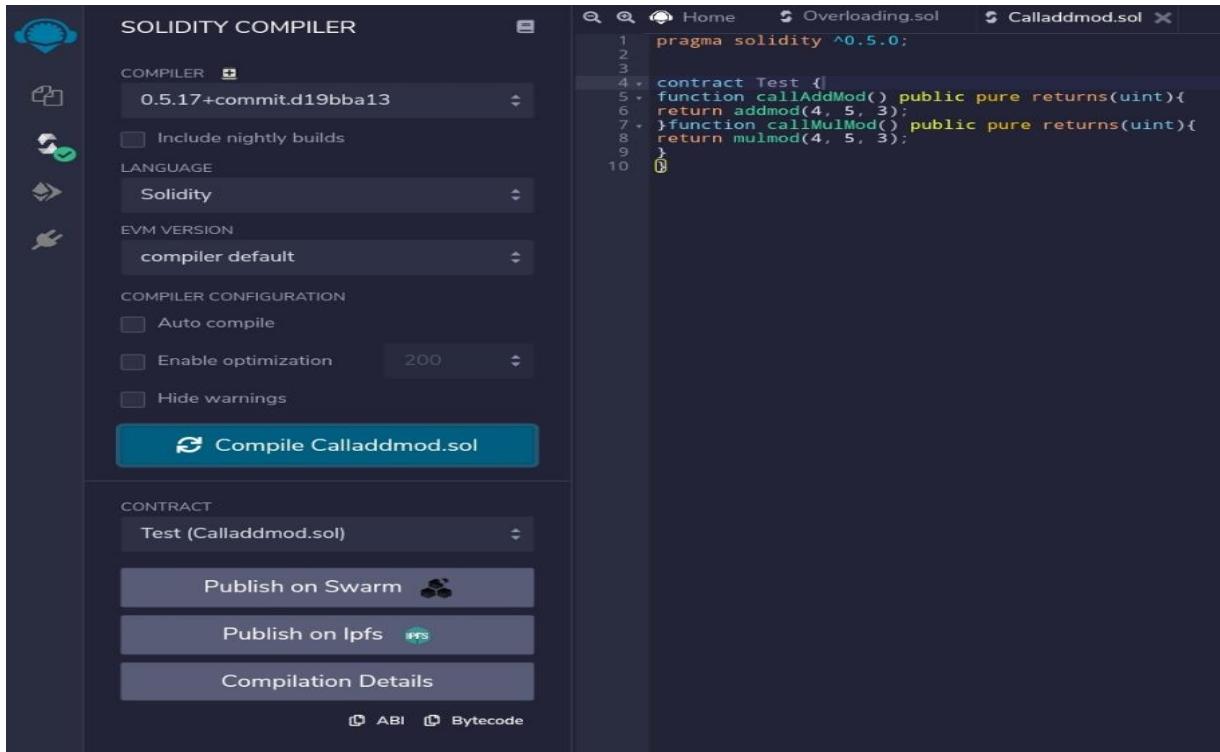
contract Test {

    function callAddMod() public pure returns(uint){
        return addmod(4, 5, 3);
    }

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

```

Output:



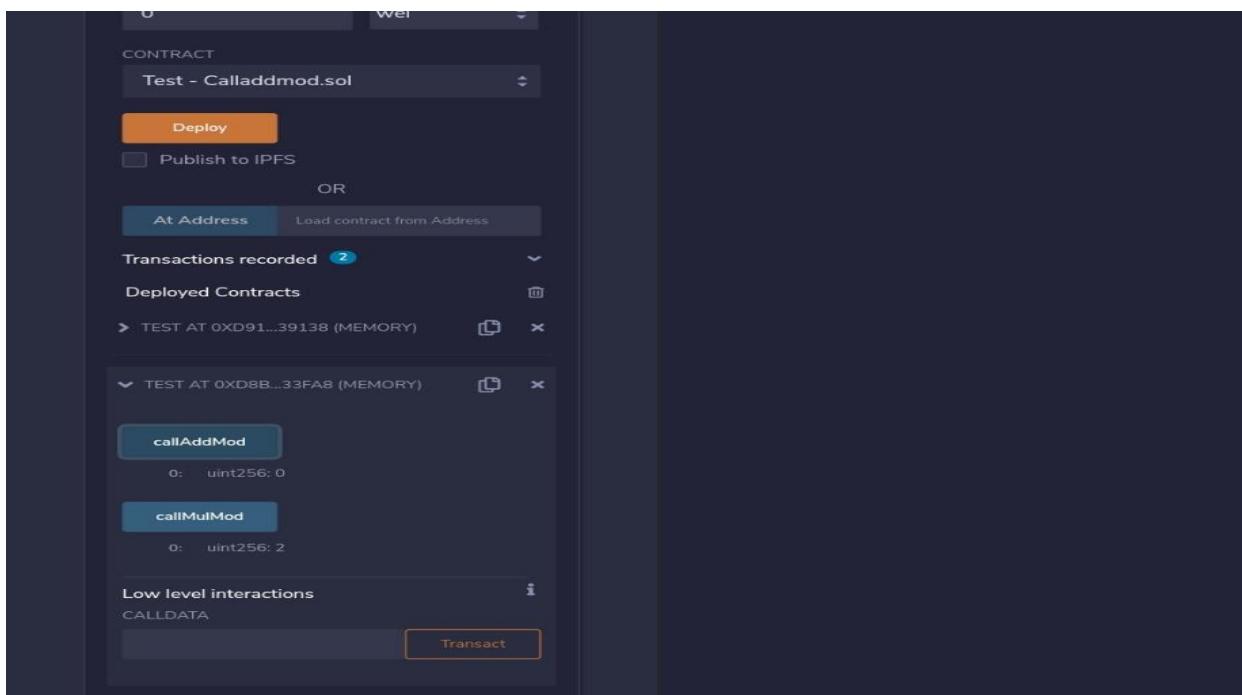
The Solidity Compiler interface shows the following configuration:

- 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) selected.

Buttons:
Compile Calladdmod.sol
Publish on Swarm
Publish on IPFS
Compilation Details
ABI
Bytecode

```
pragma solidity ^0.5.0;
contract Test {
    function callAddMod() public pure returns(uint){
        return addmod(4, 5, 3);
    }
    function callMulMod() public pure returns(uint){
        return mulmod(4, 5, 3);
    }
}
```



Contract: Test - Calladdmod.sol selected.

Deploy button is orange.

OR

At Address: Load contract from Address

Transactions recorded: 2

Deployed Contracts:

- TEST AT 0xD91...39138 (MEMORY)
- TEST AT 0xD8B...33FA8 (MEMORY)
 - callAddMod
 - 0: uint256: 0
 - callMulMod
 - 0: uint256: 2

Low level interactions:
CALLDATA
Transact

F.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 Solidity Compiler interface with the following details:

- SOLIDITY COMPILER** tab is selected.
- COMPILER**: Version 0.5.17+commit.d19bba13 is selected.
- LANGUAGE**: Solidity is selected.
- EVM VERSION**: compiler default is selected.
- COMPILER CONFIGURATION**: Auto compile is checked, Enable optimization is set to 200, and Hide warnings is checked.
- Contract**: Test (cryptodata.sol) is selected.
- Buttons**: Publish on Swarm, Publish on IPFS, and Compilation Details.
- Code Area**: Shows the Solidity code for cryptodata.sol.
- Transaction Area**: Shows a transaction with:
 - from: 0x5B38Da6a701c568545dCfcB03FcB875f56be00C4
 - to: SolidityTest.(constructor)
 - gas: 3000000
 - transaction cost: 116859 gas

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons for deployment, monitoring, and testing. The main area has two tabs: "overloading.sol" and "cryptodata.sol". The "overloading.sol" tab contains the following Solidity code:

```
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");
    }
}
```

Below the code, it says "0: uint256 9". Under "Low level interactions", there's a "CALLDATA" section with a "Transact" button. A modal window titled "TEST AT 0x332_D4B6D (MEMORY)" is open, showing results for "callkeccak256" and "callsha256".

The "callkeccak256" result is:

```
0: bytes32: result 0x2ca96055cb975b6296  
46e25b4c09bd530fb8dffa92358b0679b  
ceb0e5538eed8
```

The "callsha256" result is:

```
0: bytes32: result 0xe24dd2210803b47373  
9bd9e3163a4ca807b63201c1bc32b68fb  
122ca52eff36
```

Under "Low level interactions", there's another "CALLDATA" section with a "Transact" button. At the bottom, there's a transaction history table:

	from	to	gas
0	0x5838da6a701c568545dCfc887f596eddC4	SolidityTest.(constructor)	3000000 gas

There's also a search bar at the top right: "Search with transaction hash or address".

Practical 7

Implement and demonstrate the use of the following in Solidity:

- a) Contracts
- b) Inheritance
- c) Constructors
- d) Abstract Class
- e) Interfaces

A. 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 are several icons: a blue headphones icon, a white document icon, a green circular icon with a person, a diamond icon, a ribbon icon, a gear icon, and a flame icon. The main area has tabs for "overloading.sol", "cryptodata.sol", and "smartcontract.sol". The "smartcontract.sol" tab is active. The code editor contains the following Solidity code:

```

1 // Solidity program to
2 // demonstrate how to
3 // write a smart contract
4 pragma solidity >= 0.4.16 < 0.7.0;
5
6 // Defining a contract
7 contract Test {
8
9 // Declaring state variables
10 uint public var1;
11 uint public var2;
12 uint public sum;
13
14 // Defining public function
15 // that sets the value of
16 // the state variable
17 function set(uint x, uint y) public
18 {
19     var1 = x;
20     var2=y;
21     sum=var1+var2;
22 }
23
24 // Defining function to
25 // print the sum of
26 // state variables
27 function get(
28 ) public view returns (uint) {
29     return sum;
30 }
31 }
32
33

```

Below the code editor, there are sections for "COMPILER", "CONTRACT", and "TRANSACTION". The "TRANSACTION" section shows the transaction details:

- from: 0x58380a6a701c568545dfcB03Fc8875f56bebddC4
- to: SolidityTest.(constructor)
- gas: 3000000 gas

The screenshot shows the Deploy & Run Transactions interface. On the left, there are icons: a blue headphones icon, a white document icon, a green circular icon with a person, a diamond icon, a ribbon icon, a gear icon, and a flame icon. The main area has tabs for "overloading.sol", "cryptodata.sol", and "smartcontract.sol". The "smartcontract.sol" tab is active. The code editor contains the same Solidity code as the previous screenshot.

In the "TEST AT 0x5E1...4EFF5 (MEMORY)" section, the "set" function is selected. The parameters are set to x: 20 and y: 15. A button labeled "transact" is highlighted in orange. Below the code editor, the transaction status is shown:

- [vm] from: 0x5B3...eddC4 to: Test.set(uint256,uint256) 0x5e1...4Eff5 value: 0 wei data: 0xlab...0000f logs: 0 hash: 0x85f...143c6
- status: true Transaction mined and execution succeed
- transaction hash: 0xd5Fc5d0819161e66c32b59ab8e9c427b740d75c05ea17394c2854b3143c6

B.Inheritance

```
// Solidity program to demonstrate Single Inheritance
pragma solidity >=0.4.22 <0.6.0;

// Defining contract
contract parent{

// Declaring internal state varaiable
uint internal sum;

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

}

// Defining child contract
contract child is parent{

// Defining external function to return value of internal state variable sum
function getValue() external view returns(uint) {
    return sum;
}

}

// Defining calling contract
contract caller {

// Creating child contract object
child cc = new child();

// Defining function to call setValue and getValue functions
function testInheritance() public {
    cc.setValue();
}

function result() public view returns(uint ){
    return cc.getValue();
}

}
```

Output:

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel includes fields for ENVIRONMENT (JavaScript VM), ACCOUNT (0x5B3...eddC4), GAS LIMIT (3000000), and VALUE (0 wei). The CONTRACT dropdown is set to 'child - Hierarchical.sol'. A 'Deploy' button is visible. Below it, there's a checkbox for 'Publish to IPFS' and options for 'At Address' or 'Load contract from Address'. The 'Transactions recorded' section shows 26 entries under 'Deployed Contracts', including various test and ledger balance contracts. On the right, the code editor displays the Solidity source code for 'Hierarchical.sol':

```
// Solidity program to
// demonstrate
// Single Inheritance
pragma solidity >=0.4.22 <0.6.0;

// Defining contract
contract parent{
    // Declaring internal
    // state variable sum
    uint internal sum;
}

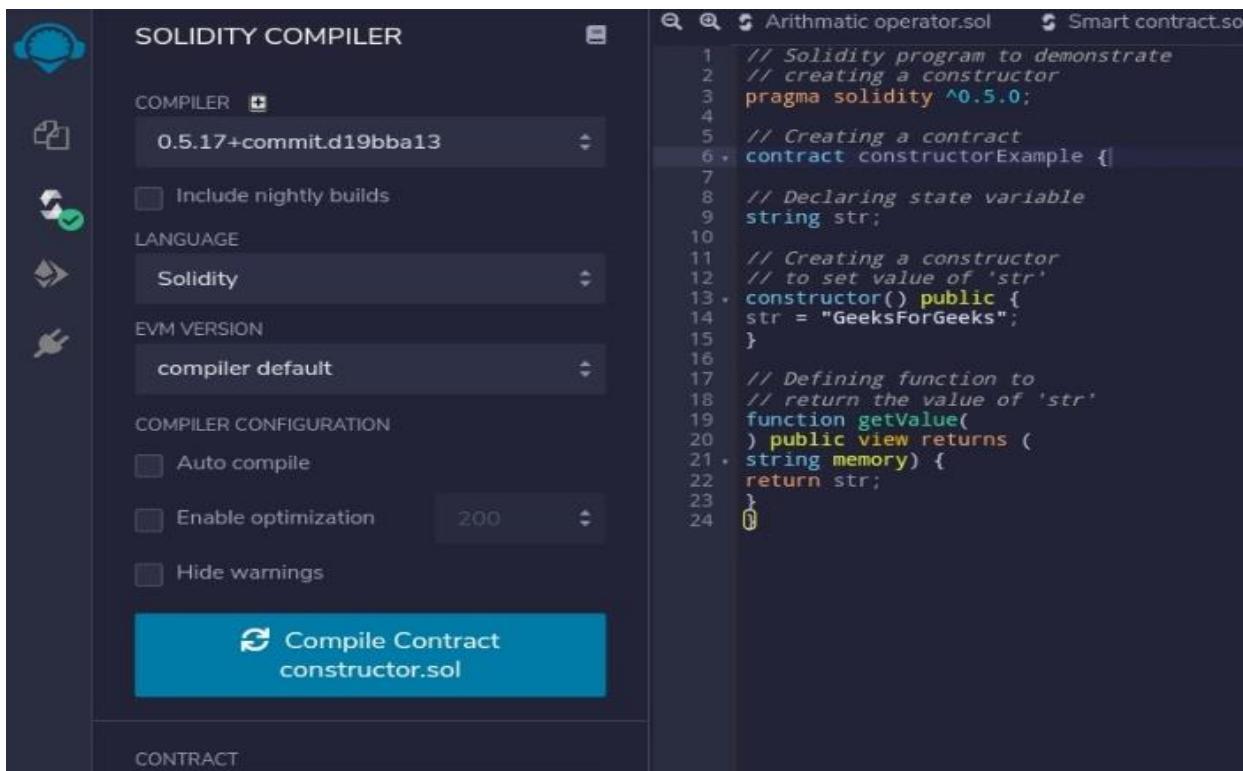
// Defining external function
// to set value of internal
// state variable sum
function setValue() external {
    uint a = 10;
    uint b = 20;
    sum = a + b;
}

// 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;
    }
}
```

C.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 Solidity Compiler interface shows the following configuration:

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

A large blue button at the bottom says "Compile Contract constructor.sol".

The right pane displays the Solidity source code:

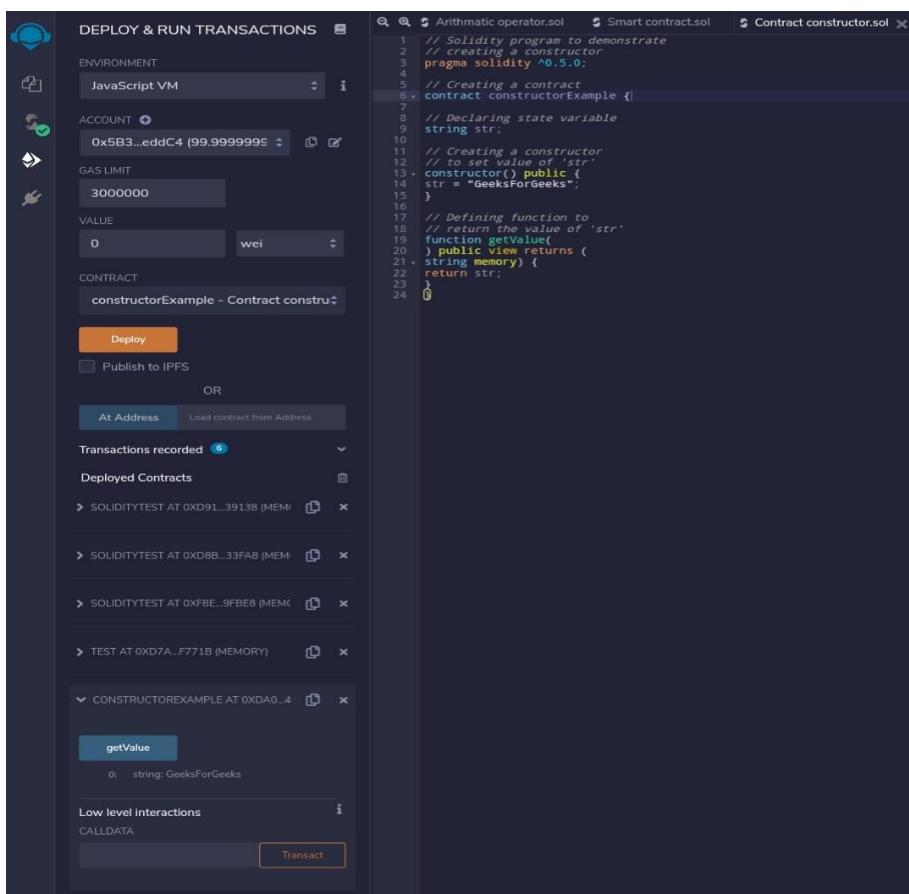
```
// Solidity program to demonstrate
// creating a constructor
pragma solidity ^0.5.0;

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

- Environment: JavaScript VM
- Account: 0x5B3...eddC4 (99.9999999)
- Gas Limit: 3000000
- Value: 0 wei
- Contract: constructorExample - Contract constru*

A large orange "Deploy" button is visible.

The right pane displays the Solidity source code and the deployed contracts:

```
// 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 Deployed Contracts section lists several deployed test contracts, including:

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

Under the CONSTRUCTOREXAMPLE contract, the "getValue" function is shown with a result of "string: GeeksForGeeks".

Practical 8

Implement and demonstrate the use of the following in Solidity:

- a) Libraries
- b) Assembly
- c) Events
- d) Error Handling

a) Libraries

// 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 screenshot shows the Solidity Compiler interface. On the left, there are several icons: a blue gear, a green checkmark, a yellow exclamation mark, a red minus sign, a blue plus sign, a green checkmark with a gear, and a blue gear with a checkmark. The main area has tabs for Home and requirestatement.sol. The code editor displays 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;
    }
}
```

Below the code editor, there is a large teal button with a circular arrow icon and the text "Compile requirestatement.sol". At the bottom, a brown bar displays the message "No Contract Compiled Yet".

The screenshot shows the Truffle UI interface. On the left, the main panel displays the 'DEPLOY & RUN TRANSACTIONS' section. 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 showing 'requireStatement - requirestatement.sol'. Below these are buttons for 'Deploy' (highlighted in orange) and 'Publish to IPFS'. The 'Transactions recorded' section shows two recent transactions: 'checkInput' at address 0xD91...3913 with value 254, and 'Odd' at the same address with value 253. The right panel shows the Solidity source code for the 'requireStatement' contract:

```
// 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;
    }
}
```

b) Assembly

```
// 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. On the left, there are several icons: a brain, a file, a green checkmark, a hand, and a gear. The main area has tabs for "SOLIDITY COMPILER" and "DEPLOY & RUN TRANSACTIONS".

SOLIDITY COMPILER:

- Compiler: 0.5.17+commit.d19bba13
- Include nightly builds:
- LANGUAGE: Solidity
- EVM VERSION: compiler default
- COMPILER CONFIGURATION:
 - Auto compile:
 - Enable optimization: 200
 - Hide warnings:
- Buttons: "Compile assertstatement.sol" (blue), "ABI", "Bytecode".
- CONTRACT: assertStatement (assertstatement.sol)
 - Publish on Swarm
 - Publish on Ipfs
 - Compilation Details

DEPLOY & RUN TRANSACTIONS:

- ENVIRONMENT: JavaScript VM
- ACCOUNT: 0x5B3...eddC4 (99.999999E)
- GAS LIMIT: 3000000
- VALUE: 0 wei
- CONTRACT: assertStatement - assertstatement.sol
 - Deploy
 - Publish to IPFS
- OR
- At Address: Load contract from Address
- Transactions recorded: 1
- Deployed Contracts: ASSERTSTATEMENT AT 0xD91...39138
 - checkOverflow: 24
 - getResult
- Low level interactions: CALLDATA
- Buttons: "Transact"

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

```
// 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 _num1, uint8 _num2) public {
        uint8 sum = _num1 + _num2;
        assert(sum<=255);
        result = true;
    }

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

c) Events

```
// Solidity program to demonstrate assert statement

pragma solidity ^0.5.0;

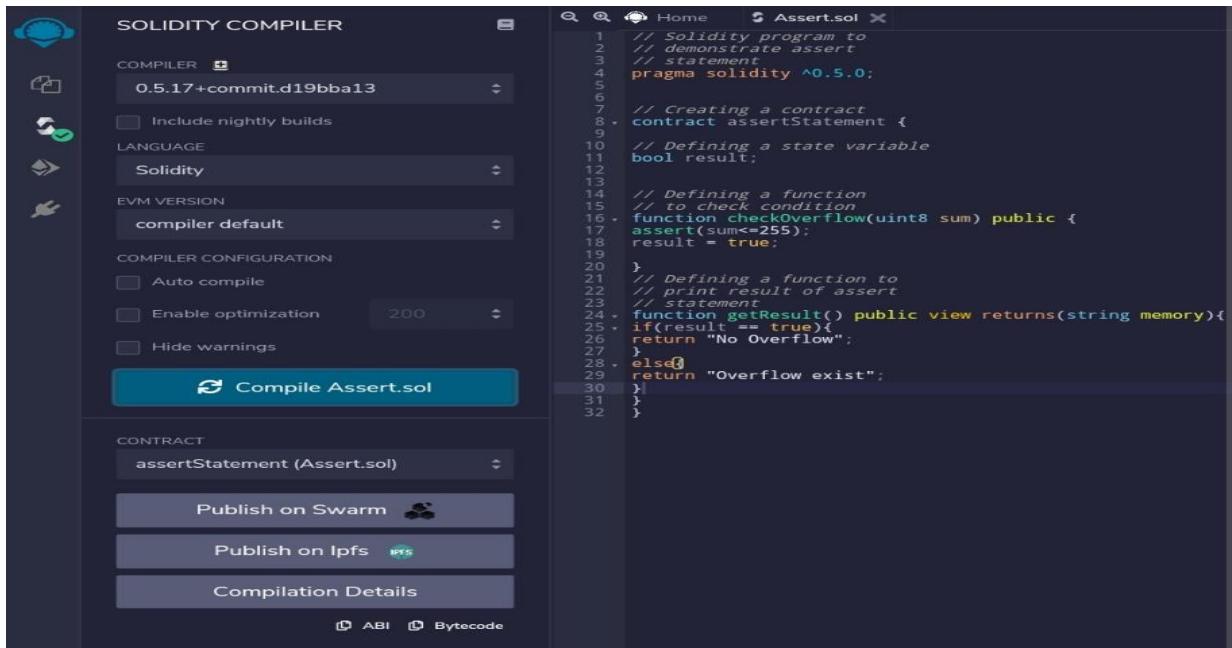
// Creating a contract
contract assertStatement {

    // 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 Solidity Compiler interface shows the following configuration for the file `Assert.sol`:

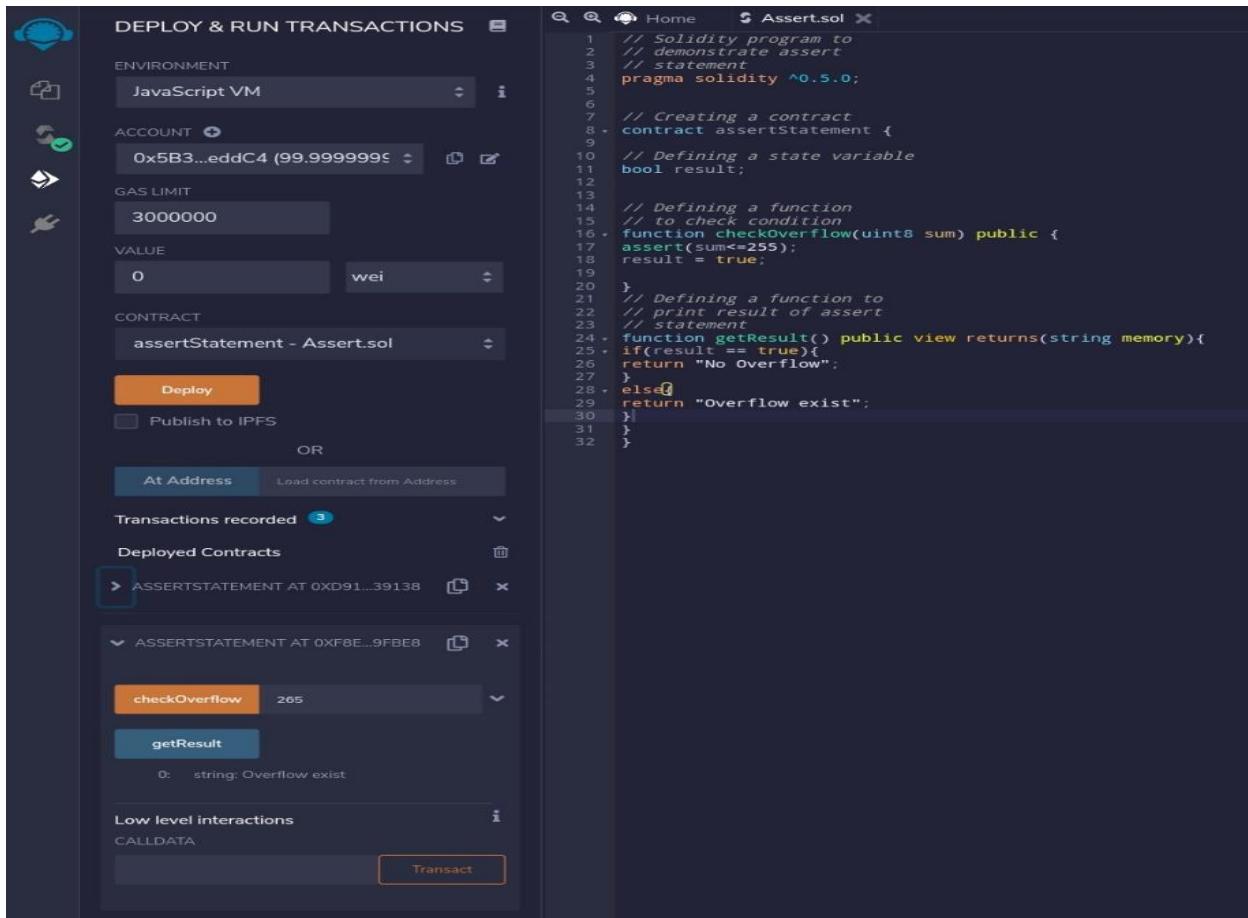
- Compiler: 0.5.17+commit.d19bba13
- Include nightly builds:
- Language: Solidity
- EVM Version: compiler default
- Compiler Configuration:
 - Auto compile:
 - Enable optimization: 200
 - Hide warnings:
- Contract: assertStatement (`Assert.sol`)
- Buttons:
 - Publish on Swarm
 - Publish on IPFS
 - Compilation Details
- ABI and Bytecode links

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



The Deploy & Run Transactions interface shows the following configuration for deploying the `assertStatement` contract:

- Environment: JavaScript VM
- Account: 0x5B3...eddC4 (99.99999999999999)
- Gas Limit: 3000000
- Value: 0 wei
- Contract: assertStatement - `Assert.sol`
- Buttons:
 - Deploy
 - Publish to IPFS
- OR
- At Address: `ASSERTSTATEMENT AT 0XD91...39138`
- Transactions recorded: 3
- Deployed Contracts:
 - `ASSERTSTATEMENT AT 0XF8E...9FBE8`
- Interactions for `ASSERTSTATEMENT AT 0XF8E...9FBE8`:
 - checkOverflow: 265
 - getResult
- Low level interactions: CALLDATA

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

d) Error Handling

```
// 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 Truffle UI interface. On the left, there's a sidebar with various icons and settings:

- SOLIDITY COMPILER
- COMPILER: 0.5.17+commitid19bba13
- Include nightly builds (checkbox)
- LANGUAGE: Solidity
- EVM VERSION: compiler default
- COMPILE CONFIGURATION:
 - Auto compile (checkbox)
 - Enable optimization: 200
 - Hide warnings (checkbox)
- Compile revertstatement.sol button

Below the sidebar, the CONTRACT section shows:

- revertStatement (revertstatement.sol)
- Publish on Swarm button
- Publish on IpfS button
- Compilation Details button

The main area contains tabs for `overloading.sol`, `cryptodata.sol`, `smartcontract.sol`, and `revertstatement.sol`. The `revertstatement.sol` tab is active, displaying the following Solidity code:

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

At the bottom, there's a network status bar with a green checkmark, transaction details, and a Debug button.

The screenshot shows the Truffle IDE interface with the following components:

- Deploy & Run Transactions**: A sidebar on the left containing buttons for "var1" and "var2".
- Low level interactions**: A section for "CALLDATA" with a "Transact" button.
- REVERTSTATEMENT AT 0x1C9...2B4BC**: A dropdown menu showing the contract address and a "call" button.
- checkOverflow**: A section with inputs for "_num1" (52) and "_num2" (35), and a "call" button.
- Output**: Results of the call:
 - 0: string: No Overflow
 - 1: uint256: 87
- Low level interactions**: Another section for "CALLDATA" with a "Transact" button.
- Code Editor**: The main area displaying the Solidity code for `overloading.sol`, which includes a `revert` statement to demonstrate overflow detection.
- Network Status**: A bottom bar showing "[vm] from: 0x5B3...eddC4 to: Test.set(uint256,uint256) 0x5e1...4FFF value: 0 wei data: 0x1ab...0000f logs: 0 hash: 0x85f...143c6" and a "Debug" button.