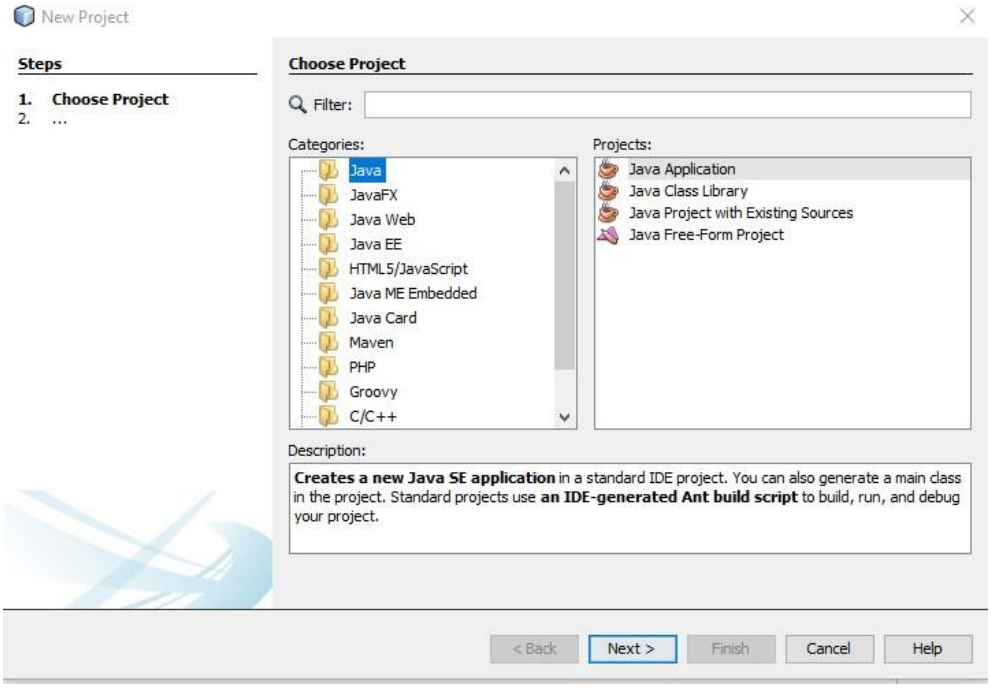
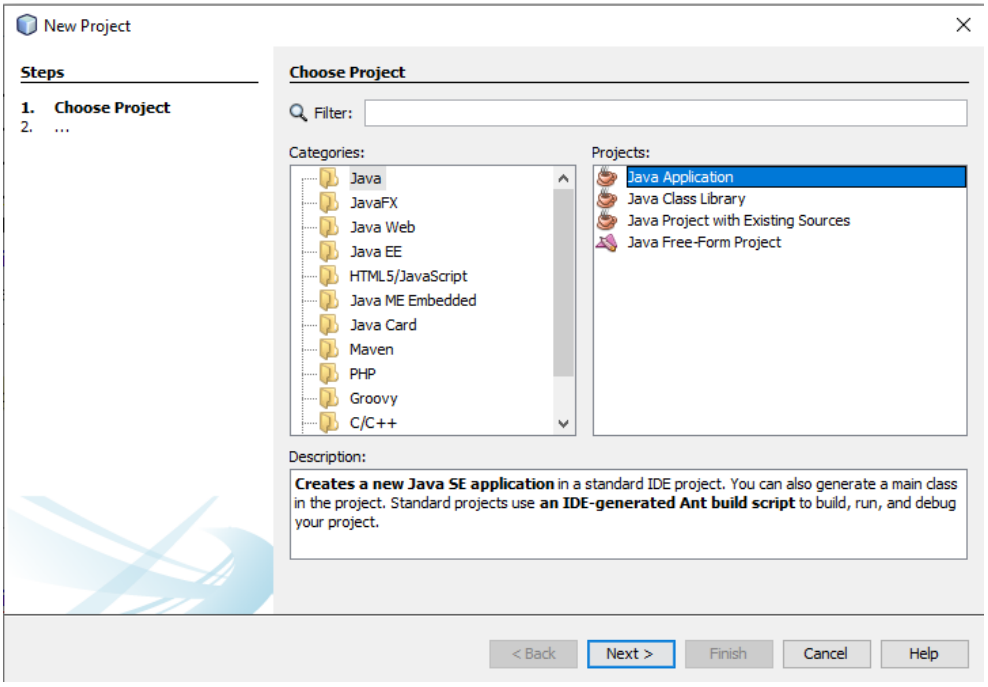
Practical No:01

Aim: 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 algorithm and tests it.

Generating keys using DSA





Code :

```
package javaapplication32;
import java.security.Keypair;
import java.security. KeyPairGenerator;
import java.security. PrivateKey;
public static void main(String args[]) throws Exception{
 KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
keypairGen.initialize (2048);
KeyPair pair = keypairGen.genKeyPair();
PrivateKey privkey = pair.getPrivate();


PublicKey publicKey = pair.getPublic();
System.out.println("Keys Generated");
System.out.println("PrivateKey"+privkey);
 System.out.println("PublicKey"+publicKey);
```

```java
        //Getting the public key from the key pair
        PublicKey publicKey = pair.getPublic();

        System.out.println("Keys Generated");
        System.out.println("PrivateKey"+privKey);
        System.out.println("PublicKey"+publicKey);
    }
}
```

Output:

```
Output - JavaApplication32 (run)
run:
Keys Generated
PrivateKeysun.security.provider.DSAPrivateKey@fffa1339
PublicKeySun DSA Public Key
    Parameters:
    p:
    8f7935d9 b9aae9bf abed887a cf4951b6 f32ec59e 3baf3718 e8eac496 1f3efd36
    06e74351 a9c41833 39b809e7 c2ae1c53 9ba7475b 85d011ad b8b47987 75498469
    5cac0e8f 14b33608 28a22ffa 27110a3d 62a99345 3409a0fe 696c4658 f84bdd20
    819c3709 a01057b1 95adcd00 233dba54 84b6291f 9d648ef8 83448677 979cec04
    b434a6ac 2e75e998 5de23db0 292fc111 8c9ffa9d 8181e733 8db792b7 30d7b9e3
    49592f68 09987215 3915ea3d 6b8b4653 c633458f 803b32a4 c2e0f272 90256e4e
    3f8a3b08 38a1c450 e4e18c1a 29a37ddf 5ea143de 4b66ff04 903ed5cf 1623e158
    d487c608 e97f211c d81dca23 cb6e3807 65f822e3 42be484c 05763939 601cd667
    q:
    baf696a6 8578f7df dee7fa67 c977c785 ef32b233 bae580c0 bcd5695d
    g:
    16a65c58 20485070 4e7502a3 9757040d 34da3a34 78c154d4 e4a5c02d 242ee04f
    96e61e4b d0904abd ac8f37ee b1e09f31 82d23c90 43cb642f 88004160 edf9ca09
    b32076a7 9c32a627 f2473e91 879ba2c4 e744bd20 81544cb5 5b802c36 8d1fa83e
    d489e94e 0fa0688e 32428a5c 78c478c6 8d0527b7 1c9a3abb 0b0be12c 44689639
    e7d3ce74 db101a65 aa2b87f6 4c6826db 3ec72f4b 5599834b b4edb02f 7c90e9a4
    96d3a55d 535bebfc 45d4f619 f63f3ded bb873925 c2f224e0 7731296d a887ec1e
    4748f87e fb5fdeb7 5484316b 2232dee5 53ddaf02 112b0d1f 02da3097 3224fe27
    aeda8b9d 4b2922d9 ba8be39e d9e103a6 3c52810b c688b7e2 ed4316e1 ef17dbde

    y:
    59a29e0e 3dee7aa8 dc0526c9 6130416a 3870485c 7dfef447 c34da309 c046a809
    4387f933 30fcf982 c3335f0e 961baa31 6b031780 9247eacb a6786fa0 77e3d98f
    5c23b611 d945bf27 b12cfd1b 37c77318 a1a101c4 7bf04ee6 0d3db4e5 ab10b610
    d7b1c74f 771e3e24 ec27c5d3 8d6e781a c023ed70 c247109f 83dcdf60 8e4c2b83
    4be82d4e b96b2ed0 36d32a4f d66ad10e dd5708db 0dc16d78 0cfe810f bea023f0
    16ad22c4 299d1ee8 e5471991 1281f40e f6985403 ccce39c9 be111f66 11ac7d9e
    0040df2d f66a7df2 aafe1547 f375d184 10fc2ff4 0e470c1d a591806f c03e7c4c
    882f51da 3bd8be22 f0d2a319 e5cb695b 6ec24fc2 f8eef769 9143ce2e bf5139e7

BUILD SUCCESSFUL (total time: 0 seconds)
```

Generating keys using RSA

[1]:
```
pip install pycryptodome
```

```
Requirement already satisfied: pycryptodome in c:\users\admin\anaconda3\lib\site-packages (3.20.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

class RSAKeyGenerator:
    def __init__(self,key_size=2048):
        self.key_size=key_size
    def generate_keys(self):
        #Generate RSA key pair
        key_pair = RSA.generate(self.key_size)

        #Extract public and private key
        public_key=key_pair.publickey().export_key()
        private_key=key_pair.export_key()
        return private_key,public_key
```

```
class RSAClient:
    def __init__(self):
        self.key_generator=RSAKeyGenerator()
    def encrypt_message(self,public_key,message):
        #Load public key
        recipient_key=RSA.import_key(public_key)

        #create cipher object
        cipher_rsa=PKCS1_OAEP.new(recipient_key)

        #Encrypt the message
        encrypted_message=cipher_rsa.encrypt(message.encode())
        return encrypted_message
    def decrypt_message(self,private_key,encrypted_message):
        #Load private key
        private_key=RSA.import_key(private_key)

        #create cipher object
        cipher_rsa=PKCS1_OAEP.new(private_key)

        #Decrypt the message
        decrypted_message=cipher_rsa.decrypt(encrypted_message)
        return decrypted_message.decode()
```

```python
#Example usage
if __name__ == "__main__":
    client=RSAClient()

    #Generate keys
    private_key,public_key=client.key_generator.generate_keys()

    #Encrypt and decrypt a message
    message="Hello, this is a test message!"
    encrypted_message=client.encrypt_message(public_key,message)
    decrypted_message=client.decrypt_message(private_key,encrypted_message)

    print("Original Message :", message)
    print("Encrypted Message :", encrypted_message)
    print("Decrypted Message :", decrypted_message)
```

Code :-

```
[1]: pip install pycryptodome
 [2]: from Crypto. PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
class RSAKeyGenerator:
def __init__(self,key_size=2048):
self.key_size-key_size
def generate_keys(self):
#Generate RSA key pair
key_pair - RSA.generate(self.key_size)
#Extract public and private key
public_key-key_pair.publickey().export_key()
private_key-key_pair.export_key() return private_key,public_key
class RSAClient:
def __init__(self):
self.key_generator-RSAKeyGenerator()
def encrypt_message(self, public_key, message): #Load public key
recipient_key=RSA. import_key (public_key)
#create cipher object
cipher_rsa=PKCS1_OAEP.new(recipient_key)
#Encrypt the message
encrypted_message=cipher_rsa. encrypt (message.encode()) return encrypted_message
def decrypt_message (self, private_key, encrypted_message): #Load private key
private_key-RSA. import_key (private_key)
#create cipher object
cipher_rsa-PKCS1_OAEP.new(private_key)
#Decrypt the message
decrypted_message=cipher_rsa.decrypt (encrypted_message)
 return decrypted_message.decode()

#Example usage
if __name_== "___main___":
client-RSAClient()
```

```
#Generate keys
private_key,public_key-client.key_generator.generate_keys()
#Encrypt and decrypt a message
message="Hello, this is a test message!"
encrypted_message=client. encrypt_message (public_key, message) decrypted_message-client.decrypt_message
(private_key, encrypted_message)
print("Original Message : message)
print("Encrypted Message:" encrypted message)
print("Decrypted Message:", decrypted_message)
```

Output:

```
Original Message : Hello, this is a test message!
Encrypted Message : b'\xa1x\x13+\xf8?$8\x19\x08\x91k\x03\x95\x7fE\x02\xa4\xadI\xef\xd0v\x92um\x05\xe8\x8a\xa0!Y3\xd5\r\xf7{\x8cK\x91\xb2h\xe7\x97\x07\xd2
\n+v\xe5!#\x07\x8c\xb0\x06\xf5&\x1aB\xe7\xc9]q\xaf\x9c\xa8!S\xc8\x1c\x1cJ\xc1wE\x95\x86w\xda%\x05\xe1\xc6JIh\xdaY\x7f\x94M)\x0ex\xc3_.\xab\xa5\xab\x9ehf
\x8c\x0ca\xd4\xf8\\\xd1<\xdb0$\x16\xe3o\xe9\xebWL\xfe\x93\x8e9\x99\xed\xfa1\xd8\xdbL8j\xbd@\x8c\x02\x92\x1f\xaf\x000%\x8dj7\x0894\x1b@}\xa6\xc4/Sv<\x01+
\xcaP(\xe3\xf3<3\xc9\xeeh\x8fln\x88\x03%\x87q\x90b\x0cr\x9e\xd6}\xd0\x96E\xf8\xc2\xa1`sSs\x93\xaa\x9d\xe4\x18_\xb1o\xea7\xbd~\xe4\xa8s\x9emJ\xf6\xaeb\x14
\xdc\x12 <%.k\xea\xc1\xce`m\xc7\xdd\x92;f\xc7\xa7"\x9f\x88\x0e\xc9^j\x99!j\x7f\x8c\xc8\x00:J$/'
Decrypted Message : Hello, this is a test message!
```

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

jupyter Prac1-Transaction Last Checkpoint: 31 minutes ago

File Edit View Run Kernel Settings Help

Code

[4]:
```
import hashlib
import json
from time import time

class Transaction:
    def __init__(self,sender,recipient,amount):
        self.sender=sender
        self.recipient=recipient
        self.amount=amount
        self.timestamp=time()
    def to_dict(self):
        return {
            'sender':self.sender,
            'recipient':self.recipient,
            'amount':self.amount,
            'timestamp':self.timestamp
        }
    def hash_transcation(self):
        transaction_string=json.dumps(self.to_dict(),sort_keys=True)
        return hashlib.sha256(transaction_string.encode()).hexdigest()
```

```
class Blockchain:
    def __init__(self):
        self.chain=[]
        self.pending_transactions=[]
        self.create_block(previous_hash = '1')
        #create the genesis block
    def create_block(self,previous_hash):
        block= {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.pending_transactions,
            'previous_hash': previous_hash or self.hash_block(self.chain[-1])
        }
        self.pending_transactions=[]
        self.chain.append(block)
    def add_transaction(self,transaction):
        self.pending_transactions.append(transaction.to_dict())
    def hash_block(self,block):
        block_string=json.dumps(block,sort_keys=True)
        return hashlib.sha256(block_string.encode()).hexdigest()
```

```python
#Test the transaction and blockchain classes
if __name__ == "__main__":
    #create a blockchain
    blockchain=Blockchain()

    #create a transcation
    transcation=Transaction(sender="Alice",recipient="Bob",amount=10)

    #Add transaction to the blockchain
    blockchain.add_transaction(transcation)

    #create a new block
    blockchain.create_block(previous_hash=None)

    #print the blockchain
    print("Blockchain:")
    print(json.dumps(blockchain.chain,indent=4))
```

Code:-

[4]: import hashlib import json
from time import time
class Transaction:
def __init__(self, sender, recipient, amount):
self.sender-sender
self.recipient-recipient self.amount-amount
self.timestamp-time()
def to dict(self):
return {
'sender': self.sender,
'recipient': self.recipient,
'amount': self.amount,
'timestamp": self.timestamp
}
def hash_transcation(self):
transaction_string-json.dumps (self.to_dict(), sort_keys=True)
hashlib.sha256(transaction_string.encode()).hexdigest()
return
class Blockchain:
def __init__(self):
self.chain=[]
self.pending_transactions=[]
self.create_block (previous_hash = '1')
#create the genesis block
def create_block (self, previous_hash):
block= {
'index': len(self.chain) + 1,
'timestamp: time(),
'transactions: self.pending_transactions,
'previous_hash': previous_hash or self.hash_block(self.chain[-1])

```
}

self.pending_transactions=[]
self.chain.append(block)
def add_transaction(self, transaction):
self.pending_transactions.append(transaction.to_dict())
def hash_block(self, block):
block_string-json.dumps (block, sort_keys=True)
return hashlib.sha256 (block_string.encode()).hexdigest()

if __name_____main___":
#create a blockchain
blockchain-Blockchain()
#create a transcation
transcation-Transaction (sender-"Alice", recipient="Bob", amount=10)
#Add transaction to the blockchain blockchain.add_transaction (transcation)
#create a new block
blockchain.create_block (previous_hash=None)
#print the blockchain
print("Blockchain:")
print(json.dumps (blockchain.chain, indent=4))
```

Output:


```
Blockchain:
[
    {
        "index": 1,
        "timestamp": 1713587481.4855232,
        "transactions": [],
        "previous_hash": "1"
    },
    {
        "index": 2,
        "timestamp": 1713587481.4855232,
        "transactions": [
            {
                "sender": "Alice",
                "recipient": "Bob",
                "amount": 10,
                "timestamp": 1713587481.4855232
            }
        ],
        "previous_hash": "1cc0053dd7951aa968f56693148502e1a48563d5b68f2cb3c085de19da58e966"
    }
]
```

Practical No: 02

Aim: Write the following programs for Blockchain in Python

I) Create multiple transactions and display them

Multiple Transaction.ipynb

File Edit View Insert Runtime Tools Help Last saved at 09:43

+ Code + Text

```python
import hashlib
import json
from time import time
```

```python
class Transaction:
    def __init__(self,sender,recipient,amount):
        self.sender = sender
        self.recipient = recipient
        self.amount = amount
        self.timestamp = time()

    def to_dict(self):
        return {
            'sender': self.sender,
            'recipient': self.recipient,
            'amount': self.amount,
            'timestamp': self.timestamp
        }

    def hash_transaction(self):
        transaction_string = json.dumps(self.to_dict, sort_keys=True)
        return hashlib.sha256(transaction_string.encode()).hexdigest()
```

```python
class Blockchain:
    def __init__(self):
        self.chain = []
        self.pending_transactions = []

        self.create_block(previous_hash = '1')

    def create_block(self,previous_hash):
        block = {
            'index': len(self.chain)+1,
            'timestamp': time(),
            'transactions': self.pending_transactions,
            'previous_hash': previous_hash or self.hash_block(self.chain[-1])
        }
        self.pending_transactions = []
        self.chain.append(block)

    def add_transaction(self,transaction):
        self.pending_transactions.append(transaction.to_dict())

    def hash_block(self,block):
        block_string = json.dumps(block,sort_keys = True)
        return hashlib.sha256(block_string.encode()).hexdigest()
```

```
[ ]  if __name__ == "__main__":

        blockchain = Blockchain()

        transcation  = Transaction(sender="Alice",recipient="Bob",amount=10)

        blockchain.add_transaction(transcation)

        blockchain.create_block(previous_hash = None)

        transcation1  = Transaction(sender="Bob",recipient="Jhon",amount=10)

        blockchain.add_transaction(transcation1)

        blockchain.create_block(previous_hash = None)

        print("Blockchain: ")
        print(json.dumps(blockchain.chain, indent = 4))
```

Code:-

Import hashlib import json
from time import time
[] class Transaction:
def __init__(self, sender, recipient, amount):
self.sender sender
self.recipient recipient
self.amount amount
self.timestamp time()
def to dict(self):
return {
}
'sender': self.sender,
'recipient': self.recipient, 'amount': self.amount,
"timestamp": self.timestamp
def hash_transaction(self):
transaction_string = json.dumps (self.to_dict, sort_keys=True) return
hashlib.sha256(transaction_string.encode()).hexdigest()
class Blockchain:
def __init__(self):
self.chain []

```
self.pending_transactions = []
self.create_block (previous_hash='1')
def create_block(self,previous_hash): block = {
}
'index': len(self.chain)+1, 'timestamp: time(),
'transactions': self.pending_transactions,
"previous_hash': previous_hash or self.hash_block(self.chain[-1])
self.pending_transactions = []
self.chain.append(block)
def add_transaction(self, transaction):
self.pending_transactions.append(transaction.to_dict())
def hash_block(self, block):
block_string = json.dumps (block, sort_keys = True)
 return hashlib.sha256 (block_string.encode()).hexdigest()

if_name__=="__main__":
blockchain Blockchain()
transcation Transaction(sender="Alice", recipient="Bob", amount=10)
blockchain.add_transaction(transcation)
blockchain.create_block (previous_hash = None)
transcation1
Transaction (sender="Bob", recipient="Jhon", amount=18)
blockchain.add_transaction(transcation1)
blockchain.create_block (previous_hash = None)
print("Blockchain: ")
print(json.dumps (blockchain.chain, indent = 4))
```

Output:

```
Blockchain:
[
    {
        "index": 1,
        "timestamp": 1714709055.7096033,
        "transactions": [],
        "previous_hash": "1"
     {
         "index": 3,
         "timestamp": 1714709055.7096033,
         "transactions": [
             {
                 "sender": "Bob",
                 "recipient": "Jhon",
                 "amount": 10,
                 "timestamp": 1714709055.7096033
             }
         ],
         "previous_hash": "c382fda82ca2f6c0d861fc7a5da1070e19ca013deff3a80808717c8b4375fb19"
     }
 ]
```

II] Create a blockchain, a genesis block and execute it.

Gensis Block.ipynb

File Edit View Insert Runtime Tools Help Last saved at 09:43

+ Code + Text

```
import hashlib
import json
from time import time
```

```
[ ] class Blockchain:
        def __init__(self):
            self.chain = []
            self.pending_transactions = []

            self.create_block(previous_hash = '1')

        def create_block(self,previous_hash):
            block = {
                'index': len(self.chain)+1,
                'timestamp': time(),
                'transactions': self.pending_transactions,
                'previous_hash': previous_hash or self.hash_block(self.chain[-1])
            }
            self.pending_transactions = []
            self.chain.append(block)

        def hash_block(self,block):
            block_string = json.dumps(block,sort_keys = True)
            return hashlib.sha256(block_string.encode()).hexdigest()
```

```
[ ] if __name__ == "__main__":

        blockchain = Blockchain()

        print("Blockchain: ")
        print(json.dumps(blockchain.chain, indent = 4))
```

Code -:

Import hashlib import json
from time import time
[] class Blockchain:
def __init__(self): self.chain []
If
_name__
self.pending transactions = []
self.create_block(previous_hash = '1')
def create_block(self, previous_hash):
block = {

```
'index': len(self.chain)+1, 'timestamp: time(),
'transactions': self.pending_transactions,
'previous_hash": previous_hash or self.hash_block(self.chain[-1])
}

self.pending transactions = []
self.chain.append(block)
def hash_block(self, block):
block_string json.dumps (block, sort_keys = True)
return hashlib.sha256 (block_string.encode()).hexdigest()
if__name__=="__main__":
blockchain Blockchain()
print("Blockchain: ")
print(json.dumps (blockchain.chain, indent = 4))
```

Output:

```
Blockchain:
[
    {
        "index": 1,
        "timestamp": 1714709377.308939,
        "transactions": [],
        "previous_hash": "1"
    }
]
```

Practical No: 03

Aim: Implement and demonstrate the use of the following in solidity:

1) **Variable**
2) **Operator**
3) **Loops**
4) **Decision Making**
5) **Strings**

Open Remix.ide

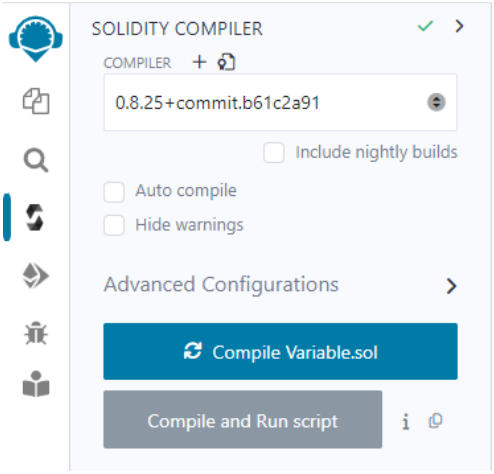
1] Variable

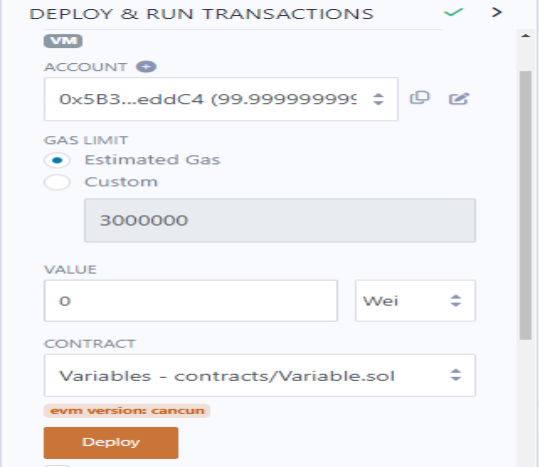
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Variables{
  uint public myNumber; //Unsigned integer variable
  address public muAddress; // Ethereum address variable
  string public myString; // string variable
  bool public mybool; //Boolean variable

  constructor() {    infinite gas 186400 gas
    myNumber = 123;
    muAddress = msg.sender;
    myString ="Hello, Solidity!";
    mybool = true;
  }
}
```

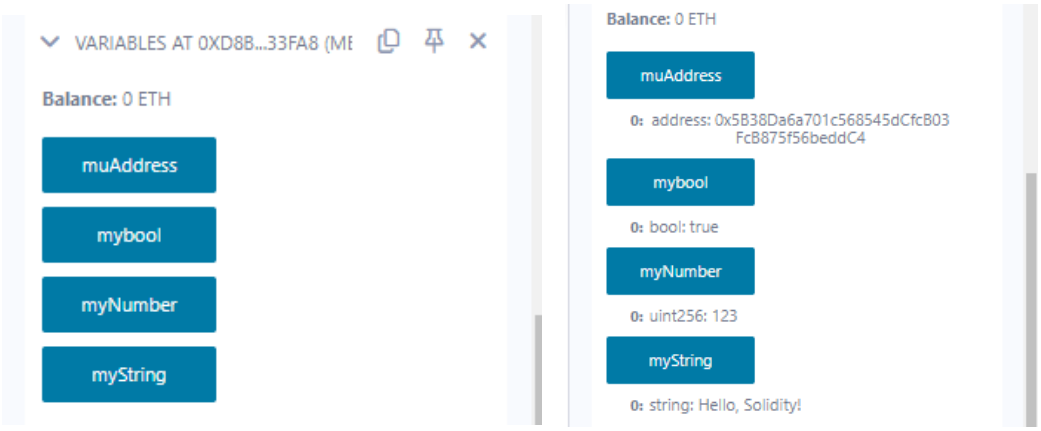
Code:

```
pragma solidity ^0.8.0;
contract Variables{
uint public myNumber
 address public muAddress;
string public myString;
bool public mybool;
constructor(){
 myNumber = 123;
muAddress msg.sender;
myString "Hello, Solidity!";
 mybool true;
}
```

Output:


SOLIDITY COMPILER
COMPILER
0.8.25+commit.b61c2a91
Include nightly builds
Auto compile
Hide warnings
Advanced Configurations
Compile Variable.sol
Compile and Run script


DEPLOY & RUN TRANSACTIONS
VM
ACCOUNT
0x5B3...eddC4 (99.99999999
GAS LIMIT
Estimated Gas
Custom
3000000
VALUE
0
Wei
CONTRACT
Variables - contracts/Variable.sol
evm version: cancun
Deploy



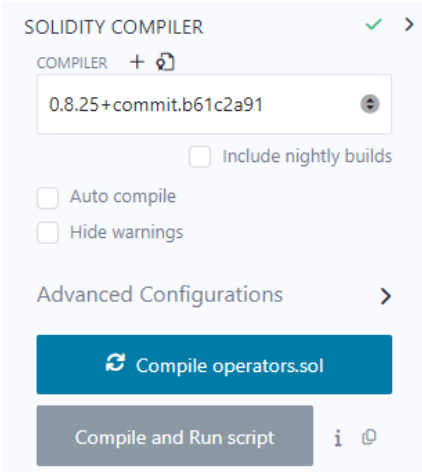
2] Operator

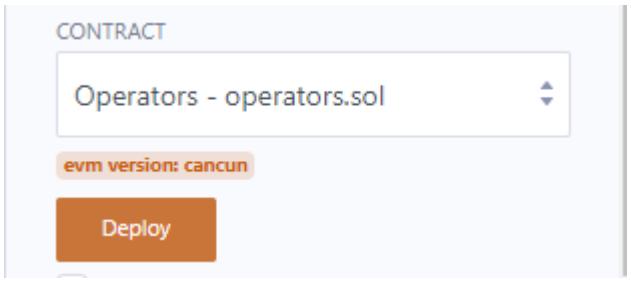
```
pragma solidity ^0.8.0;
contract Operators{
    uint public result;
    bool public ans;

    constructor() {    infinite gas 56800 gas
        uint a = 10;
        uint b = 5;


        //Arithmetic operators
        result = a + b; // Addition
        result = a - b; // Substraction
        result = a * b; // Multiplication
        result = a / b; // Division
        result = a % b; // Modulus

        // //Comparison operators
        ans = (a == b); //Equal to
        ans = (a != b); // Not Equal to
        ans = (a > b); //Greater than
        ans = (a < b); //Less than
        ans = (a >= b); //Greater than or equal to
        ans = (a <= b); //Less than or Equal to

        //Logical Operators
        ans = (a > 0 && b > 0); //Logical AND
        ans = (a > 0 || b > 0); //Logical OR
        ans = !(a > 0); //Logical NOT
    }
}
```

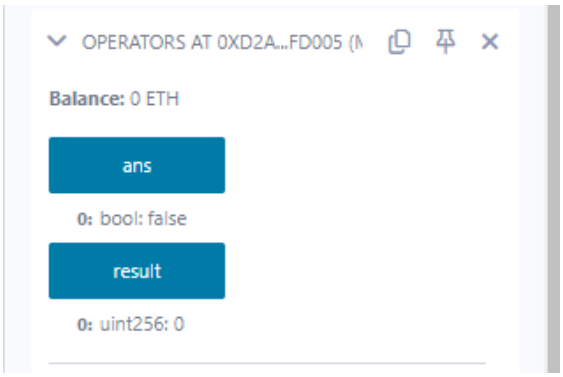
Code:-

```
pragma solidity ^0.8.0;
contract Operators{
uint public result;
bool public ans;
constructor(){
uint a = 10;
uint b = 5;
result = a + b;
result= a- b;
result =a *b
 result= a /b;
result a % b;
////Comparison operators
 ans = (a==b); //Equal to ans
ams = (a != b); // Not Equal to
ans =(a >); //Greater than
ans (a<b); //Less than
ans = (a >= b); //Greater than or equal to
ans (a <- b); //Less than or Equal to
//Logical Operators
ans = (a > 0 && b > 0); //Logical AND
 ans = (ae || b > 0); //Logical OR
 ans = !(a> 0); //Logical NOT
}
}
```

Output:


SOLIDITY COMPILER
COMPILER
0.8.25+commit.b61c2a91
Include nightly builds
Auto compile
Hide warnings
Advanced Configurations
Compile operators.sol
Compile and Run script


CONTRACT
Operators - operators.sol
evm version: cancun
Deploy


OPERATORS AT 0XD2A...FD005 (M
Balance: 0 ETH
ans
0: bool: false
result
0: uint256: 0

3] Loops

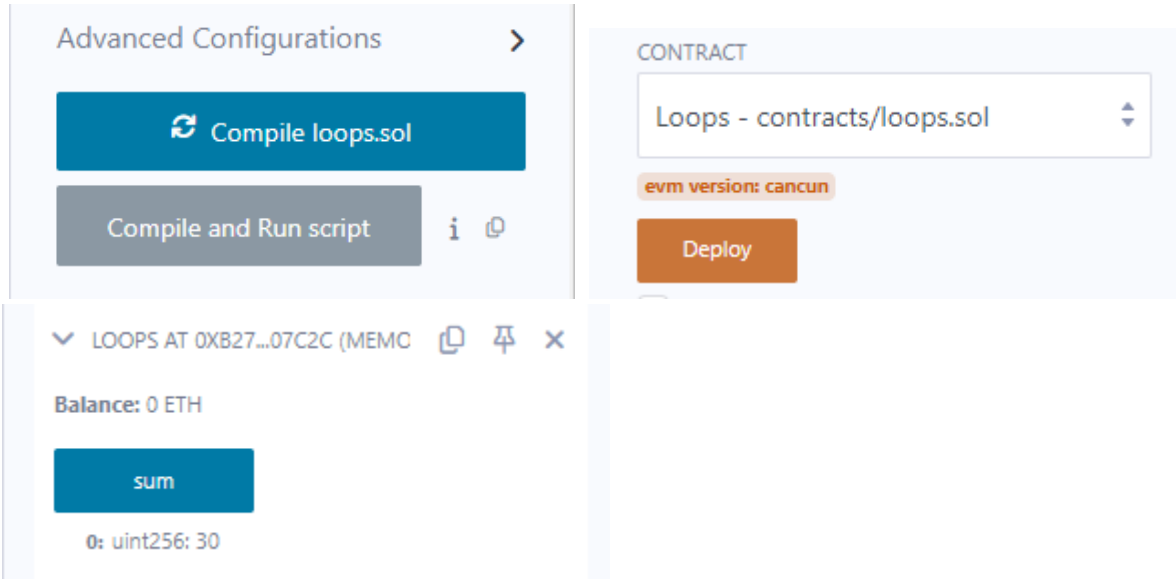
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Loops{
    uint public sum;
    constructor(){    infinite gas 35800 gas
        uint[] memory numbers = new uint[](5);
        numbers[0] = 1;
        numbers[1] = 2;
        numbers[2] = 3;
        numbers[3] = 4;
        numbers[4] = 5;

        //For loop
        for (uint i = 0; i < numbers.length; i++){
            sum += numbers[i];
        }

        //While Loop
        uint j = 0;
        while(j < numbers.length){
            sum += numbers[j];
            j++;
        }
    }
}
```

Code:-

```
pragma solidity "0.8.0;
contract Loops{
uint public sum;
constructor(){
infinite gas 35800 gas
uint[] memory numbers new uint[](5);
numbers[0] = 1;
numbers[1] = 2;
numbers[2]= 3;
numbers[3]=4;
numbers[4]= 5;
//For loop
for (uint i = 0; i < numbers.length; i++) {
sum += numbers[1];
}
//While Loop
uint j=0;
```

```
while(j < numbers.length){
sum numbers[j];
j++;
}
}
}
```

Output



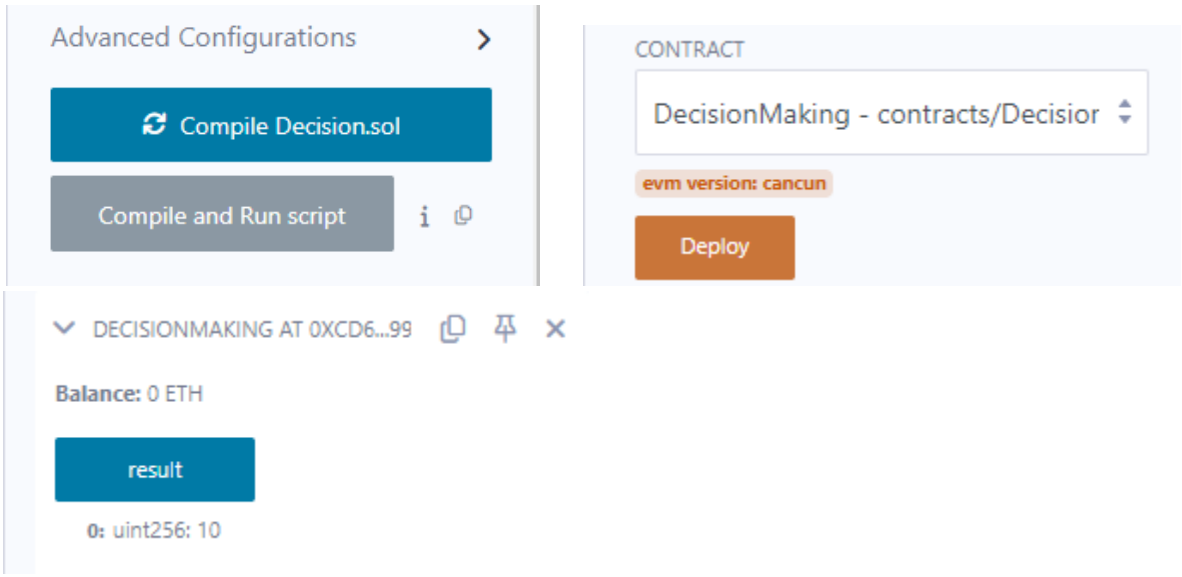
4] Decision Making

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract DecisionMaking{
    uint public result;

    constructor(){    63102 gas 35800 gas
        uint a = 10;
        uint b = 5;

        //If statement
        if(a > b){
            result = a;
        }else{
            result = b;
        }

        // Ternary operator (conditional expression)
        result = (a > b) ? a : b;
    }
}
```

Code:-

```
// SPDX-License-Identifier: MIT
pragma solidity "0.8.0;
contract Decision Making{
uint public result;
constructor(){
uint a 10;
uint b = 5;
//If statement
if(a> b){
result = a;
}else{
result = b;
}

// Ternary operator (conditional expression)
result (ab)? a: b;
}
}
```

Output:


Advanced Configurations
Compile Decision.sol
Compile and Run script
CONTRACT
DecisionMaking - contracts/Decisior
evm version: cancun
Deploy
DECISIONMAKING AT 0XCD6...99
Balance: 0 ETH
result
0: uint256: 10

5] String

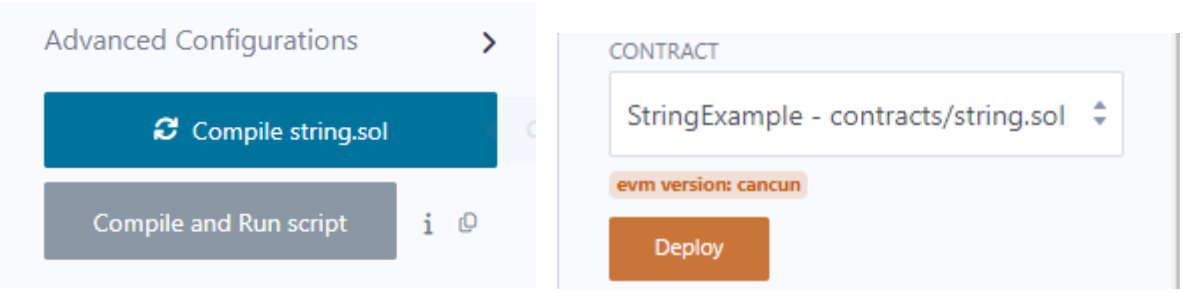
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract StringExample{
    string public myString;
    string public substring;
    bool public Comparing;
    uint public length;

    constructor(){    infinite gas 188800 gas
        //Assigning a string
        myString = "Hello, Solidity!";

        //Concatenation
        myString = string(abi.encodePacked(myString, "Welcome!"));

        //Length of string
        length = bytes(myString).length;

        //Substring extraction
        substring = substr(myString, 7, 8);

        //Comparing strings
        Comparing = compareStrings(myString, "Hello, Solidity! welcome");

    }
    function substr(string memory str, uint startIndex, uint length) internal  pure returns (string memory){    undefined gas
        bytes memory strBytes = bytes(str);
        bytes memory result = new bytes(length);
        for (uint i =0; i < length; i++){
            result[i] = strBytes[startIndex + i];
        }
    return string(result);
    }
    function compareStrings(string memory str1, string memory str2) internal  pure returns (bool){    undefined gas
        return keccak256(abi.encodePacked(str1)) == keccak256(abi.encodePacked(str2));
    }
}
```

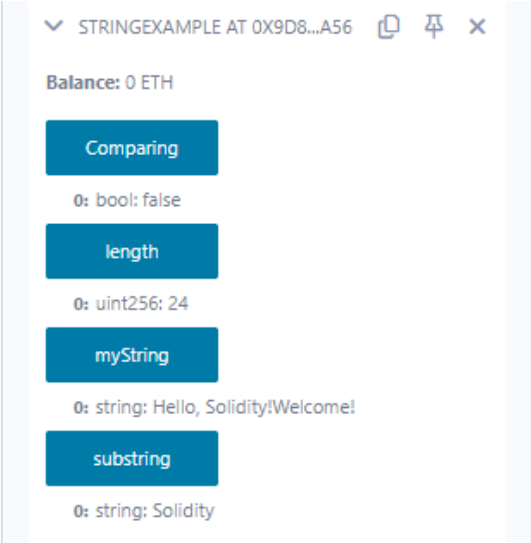
Code:-

```
pragma solidity "0.8.0;
contract StringExample{
string public myString;
string public substring;
bool public Comparing;
uint public length;
constructor(){
//Assigning a string
myString = "Hello, Solidity!";
//Concatenation
myString string (abi.encodePacked (myString, "Welcome!"));
```

```
//Length of string
length bytes (myString).length;
//Substring extraction
substring = substr(myString, 7, 8);
//Comparing strings
Comparing compareStrings (myString, "Hello, Solidity! welcome");
```

Output:




STRINGEXAMPLE AT 0X9D8...A56
Balance: 0 ETH
Comparing
0: bool: false
length
0: uint256: 24
myString
0: string: Hello, Solidity!Welcome!
substring
0: string: Solidity

Practical No:04

Aim: Implement and demonstrate the use of the following in solidity:

1] Arrays

2] Enums

3] Structs

4] Mappings

5] Ether units

6] Special Variables

1] Arrays

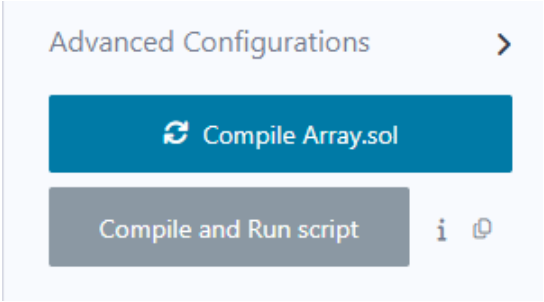
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract ArrayExample{
  uint[] public numbers;
  function addNumber(uint _number) public {    46873 gas
    numbers.push(_number);
  }

  function getAllNumbers() public view returns (uint[] memory){    infinite gas
    return numbers;
  }
  function getNumber(uint _index) public view returns (uint){    infinite gas
    require(_index < numbers.length, "Index out of bounds");
    return numbers[_index];
  }
}
```

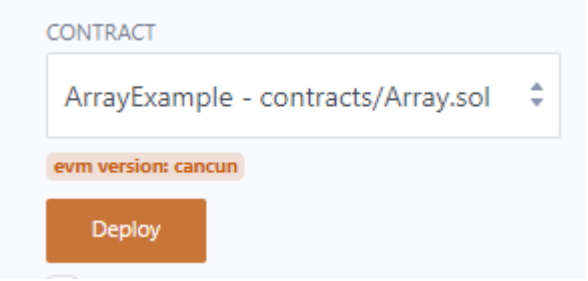
Code:-

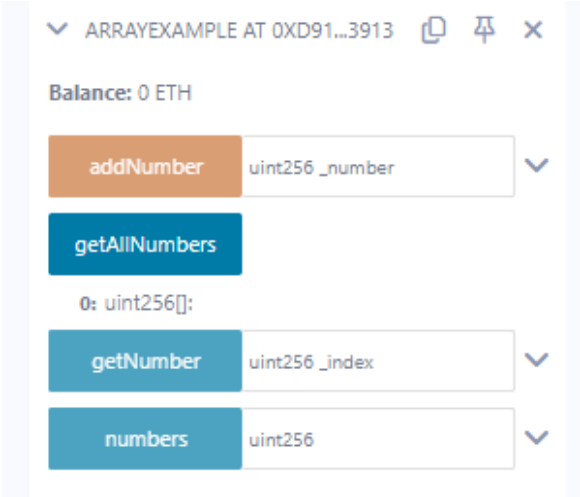
```
pragma solidity "0.8.0;
contract ArrayExample{
uint[] public numbers;
function addNumber (uint _number) public {
numbers.push(_number);
}

function getAllNumbers() public view returns (uint[] memory){
return numbers;
}
function getNumber (uint _index) public view returns (uint){
require(_index < numbers.length, "Index out of bounds");
 return numbers[_index];
}
```

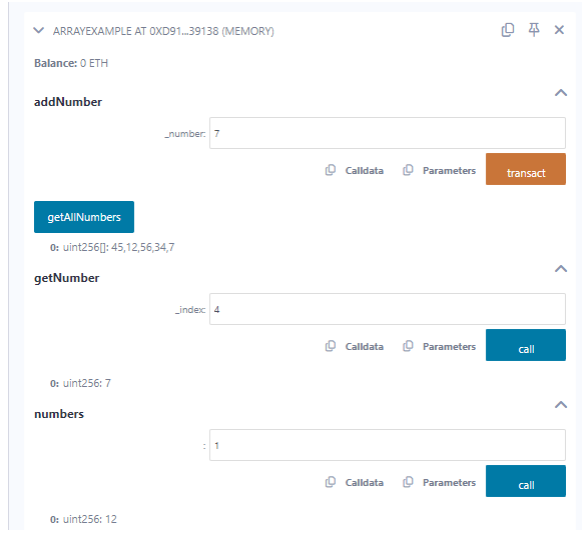
}

Output:

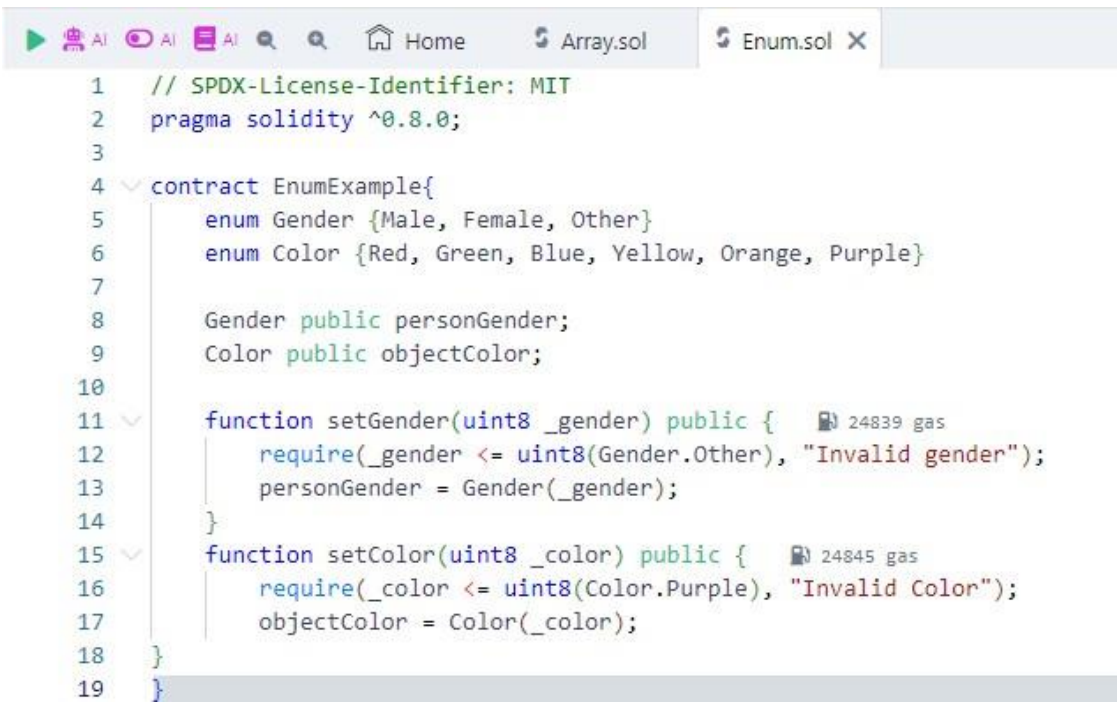

Advanced Configurations
Compile Array.sol
Compile and Run script


CONTRACT
ArrayExample - contracts/Array.sol
evm version: cancun
Deploy





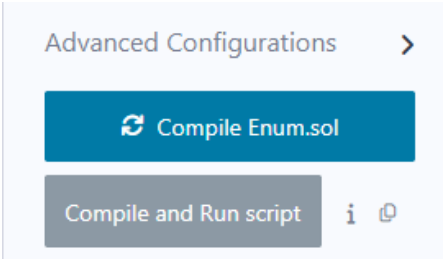
2] Enum

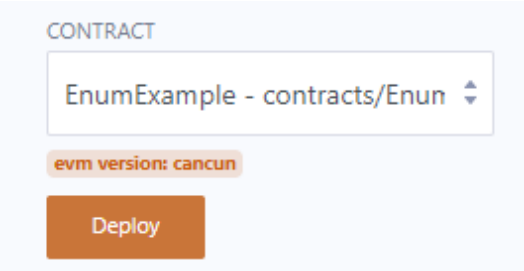


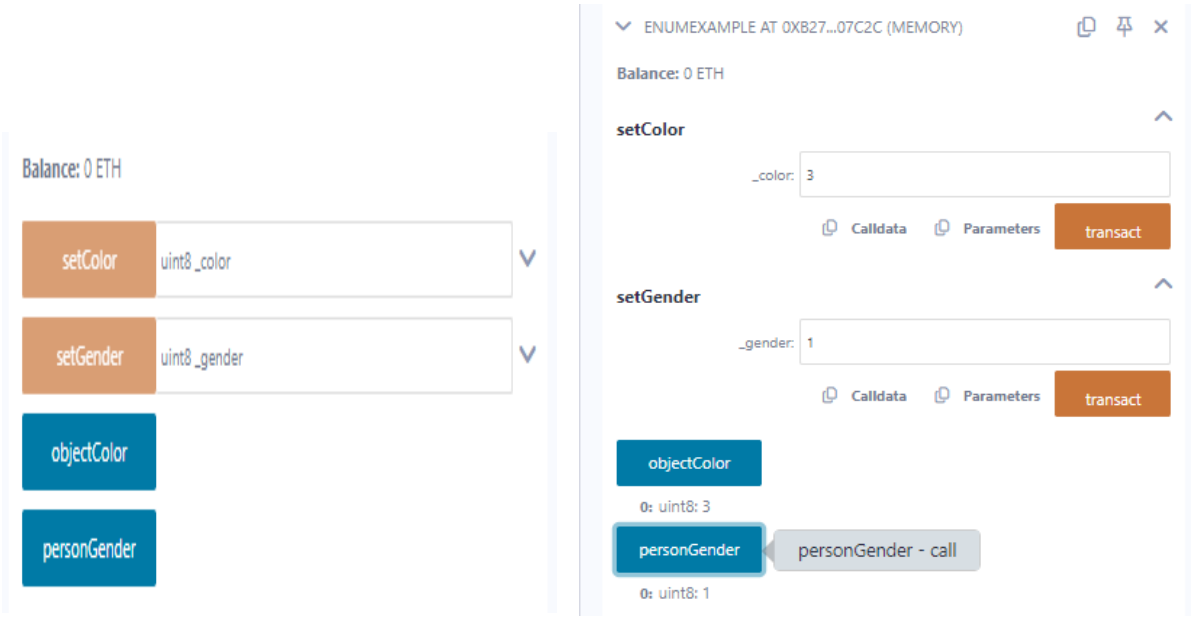
pragma solidity ^0.8.0;
contract EnumExample{
enum Gender {Male, Female, Other}
enum Color {Red, Green, Blue, Yellow, Orange, Purple}
Gender public personGender;
Color public objectColor;
function setGender (uint8_gender) public {

```
 require(_gender <= uint8 (Gender. Other), "Invalid gender");

 personGender Gender (_gender);
}
function setColor(uint8 _color) public {
require(_color<= uint8 (Color. Purple), "Invalid Color");
objectColor=Color(_color);

}
```

Output:







3] Structs

Enum.sol | Stuct.sol

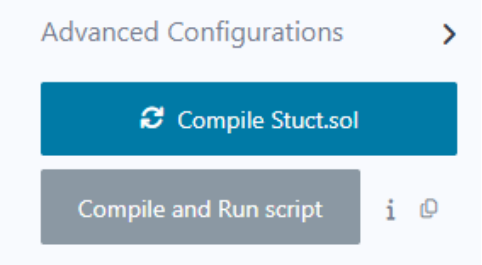
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract StructExample{
    struct Person{
        string name;
        uint age;
    }

    Person public myPerson;

    function setPerson(string memory _name, uint _age) public {    infinite gas
        myPerson = Person(_name,_age);
    }

    function getPerson() public view returns (string memory,uint){    infinite gas
        return (myPerson.name, myPerson.age);
    }
}
```

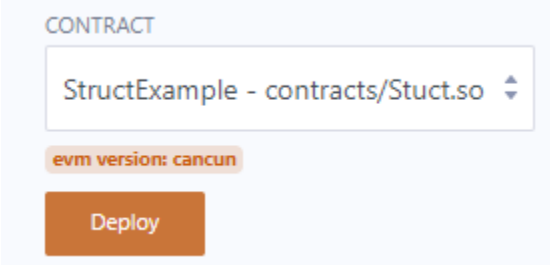
Code:-

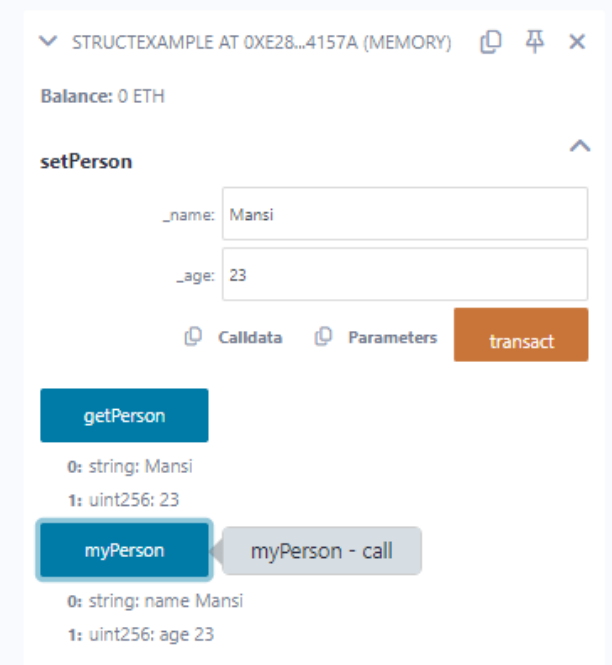
```
pragma solidity "0.8.0;
contract StructExample{
struct Person{
string name;
uint age;
}
Person public myPerson;
```

```
function setPerson(string memory _name, uint _age) public {

myPerson = Person(_name,_age);
}
function getPerson() public view returns (string memory, uint){
 return (myPerson.name, myPerson.age);
}
}
```

Output:





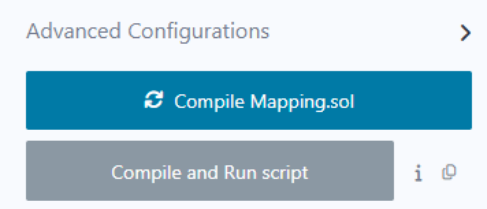


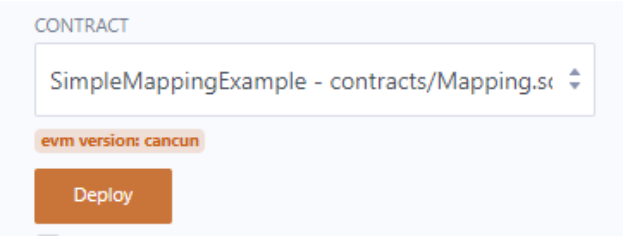
4] Mappings

Code:-

```
pragma solidity "0.8.0;
 contract SimpleMappingExample{
mapping (uint => bool) public myMapping;
function setValue(uint _index) public {
myMapping[_index] = true;
}
}
```

Output:


Advanced Configurations
Compile Mapping.sol
Compile and Run script


CONTRACT
SimpleMappingExample - contracts/Mapping.s
evm version: cancun
Deploy

SIMPLEMAPPINGEXAMPLE AT 0XC3B...AAECA (I

Balance: 0 ETH

setValue | uint256 _index

myMapping | uint256

SIMPLEMAPPINGEXAMPLE AT 0XC3B...AAECA (I

Balance: 0 ETH

setValue

_index: uint256

Calldata Parameters transact

myMapping

: 2

Calldata Parameters call

0: bool: false

SIMPLEMAPPINGEXAMPLE AT 0XC3B...AAECA (I

Balance: 0 ETH

setValue

_index: 2

Calldata Parameters transact

myMapping

: 2

Calldata Parameters call

0: bool: false

SIMPLEMAPPINGEXAMPLE AT 0XC3B...AAECA (I

Balance: 0 ETH

setValue

_index: 2

Calldata Parameters transact

myMapping

: 2

Calldata Parameters call

0: bool: true

5] Ether units

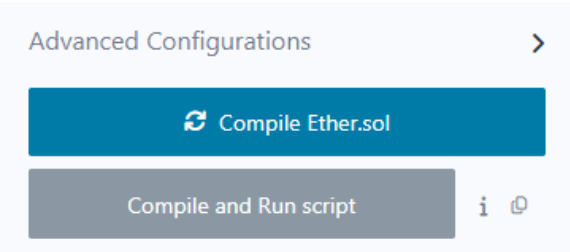
Mapping.sol Ether.sol

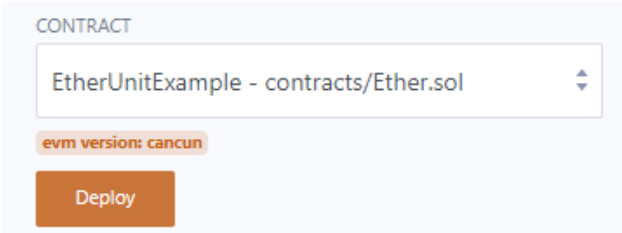
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract EtherUnitExample{
    uint256 constant KWEI_PER_WEI = 1e3;

    function convertEther(uint _valueInEther) public pure returns (uint){    infinite gas
        return _valueInEther * 1 ether;
    }
    function convertWei(uint _valueInWei) public pure returns (uint){    infinite gas
        return _valueInWei / 1 ether;
    }
    function convertToKwei(uint _valueInWei) public pure returns (uint){    infinite gas
        return _valueInWei / KWEI_PER_WEI;
    }
    function convertFromKwei(uint _valueInKwei) public pure returns (uint){    infinite gas
        return _valueInKwei * KWEI_PER_WEI;
    }
}
```

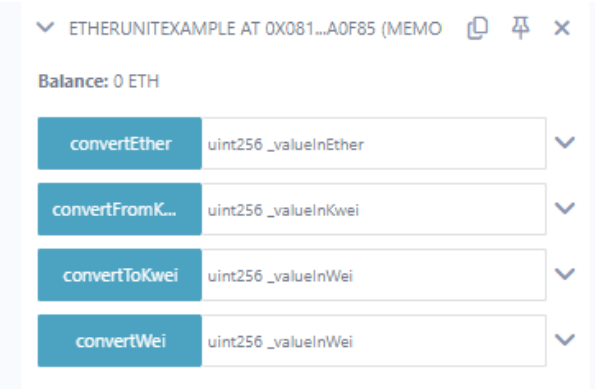
Code:-

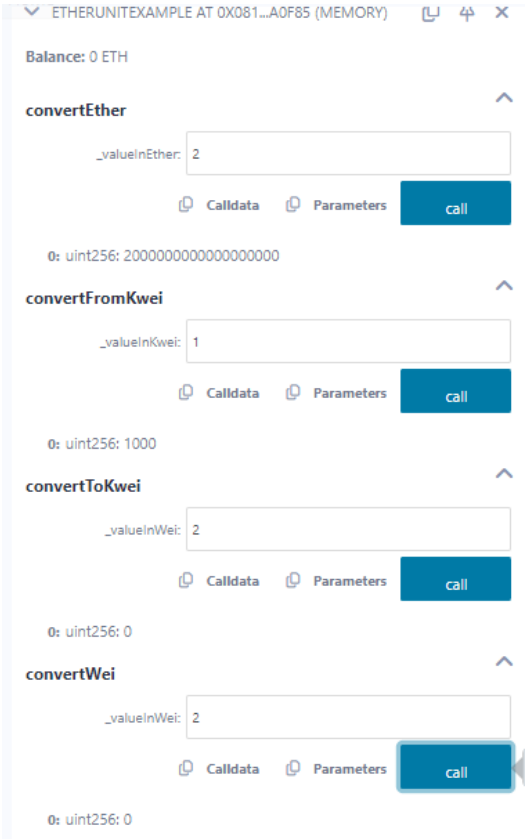
```
pragma solidity "0.8.0;
contract EtherUnitExample{
uint256 constant KWEI_PER_WEI = 1e3;
function convertEther (uint _valueInEther) public pure returns (uint){
 return _valueInEther * 1 ether;
}
function convertwei (uint value Inwei) public pure returns (uint) {
return _valueInwei / 1 ether;
}
function convertTokwei (uint valueInwei) public pure returns (uint) {
 return _valueInwei / KWEI_PER_WEI;
}
function convertFromKwei (uint _value Inkwei) public pure returns (uint) {
return _valueInKwei KWEI_PER_WEI;
}
}
```

Output:


Advanced Configurations
Compile Ether.sol
Compile and Run script


CONTRACT
EtherUnitExample - contracts/Ether.sol
evm version: cancun
Deploy


ETHERUNITEXAMPLE AT 0X081...A0F85 (MEMO
Balance: 0 ETH
convertEther
uint256 _valueInEther
convertFromK...
uint256 _valueInKwei
convertToKwei
uint256 _valueInWei
convertWei
uint256 _valueInWei



6] Special Variables

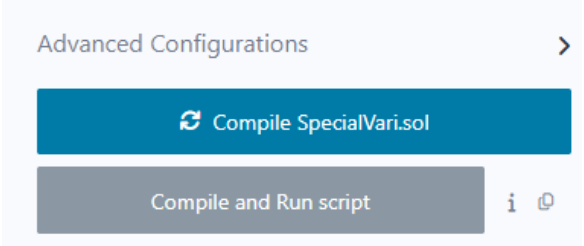
Ether.sol | SpecialVari.sol

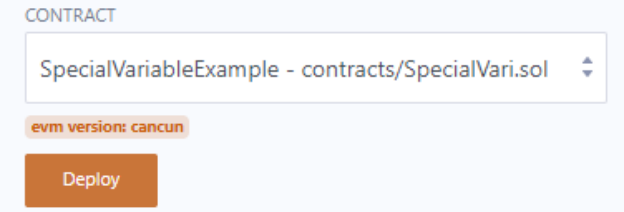
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract SpecialVariableExample{
    address public owner;

    constructor(){    142831 gas 118400 gas
        owner = msg.sender;
    }
    function getMessageSender() public view returns (address){    385 gas
        return msg.sender;
    }
    function getContractBalance() public view returns (uint){    317 gas
        return address(this).balance;
    }
    function isOwner() public view returns (bool){    2540 gas
        return msg.sender == owner;

    }
}
```

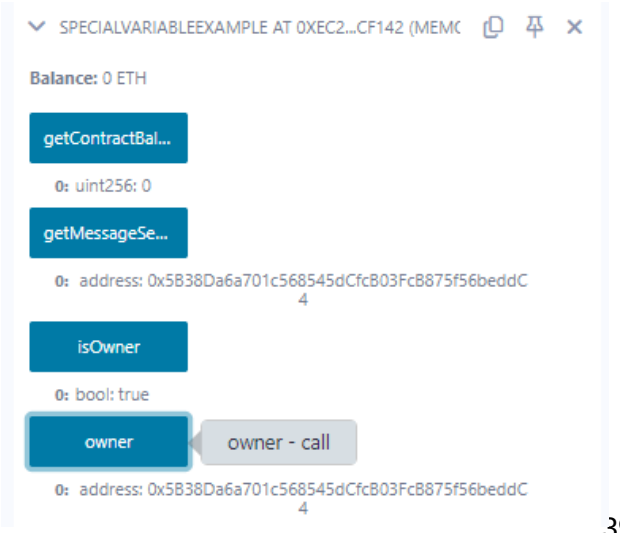
Code:-

```
pragma solidity ^0.8.0;
contract SpecialVariableExample{
address public owner;
constructor(){ 142831 gas 118400 gas
owner msg.sender;
}
function getMessageSender() public view returns (address) {
 return msg.sender;
}
function getContractBalance() public view returns (uint){
 return address (this).balance;
}
function isOwner() public view returns (bool) {
return msg.sender == owner;
}
}
```

Output:







Practical No:05

Aim: Implement and demonstrate the use of the following in solidity:

1] Functions

2] View functions

3] Pure functions

4] Fallback functions

5] Function Overloading

6] Mathematical functions

7] Cryptographic functions

1] Functions

Home | Functions.sol

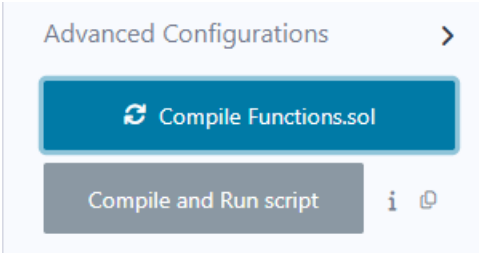
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract AddTwoValues{
  //Function to add two values
  function add(uint256 a,uint256 b) public pure returns (uint256){    infinite gas
    return a+b;
  }
   function sub(uint256 a,uint256 b) public pure returns (uint256){    infinite gas
    return a-b;
  }
   function mul(uint256 a,uint256 b) public pure returns (uint256){    infinite gas
    return a*b;
  }
   function div(uint256 a,uint256 b) public pure returns (uint256){    infinite gas
    return a/b;
  }
}
```

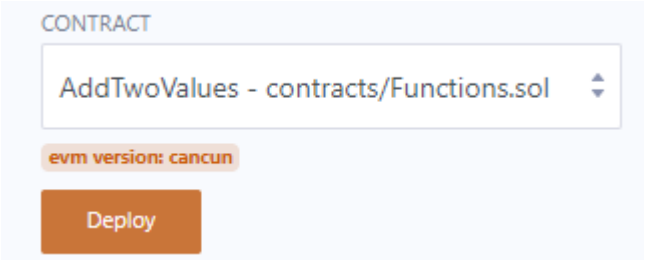
Code:-

```
pragma solidity "0.8.0;
contract AddTwoValues{
//Function to add two values
function add (uint256 a,uint256 b) public pure returns (uint256) {
return a+b;
}
function sub(uint256 a, uint256 b) public pure returns (uint256) {
 return a-b;
}
function mul (uint256 a, uint256 b) public pure returns (uint256){
 return a*b;
```

```
}
function div(uint256 a, uint256 b) public pure returns (uint256) {
return a/b;
}
}
```

Output:





ADDTWOVALUES AT 0XF8E...9FBE8 (MEI

Balance: 0 ETH

add uint256 a, uint256 b

div uint256 a, uint256 b

mul uint256 a, uint256 b

sub uint256 a, uint256 b

ADDTWOVALUES AT 0XF8E...9FBE8 (MEMORY)

Balance: 0 ETH

add

a: 5

b: 4

Calldata Parameters call

0: uint256: 9

div

a: 10

b: 5

Calldata Parameters call

0: uint256: 2

mul

a: 4

b: 2

Calldata Parameters call

0: uint256: 8

sub

a: 5

b: 2

Calldata Parameters call

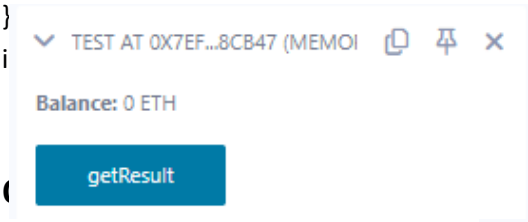
0: uint256: 3

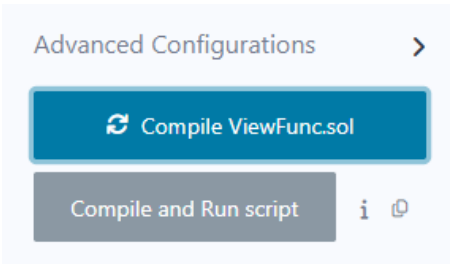
2] View functions - Read Only

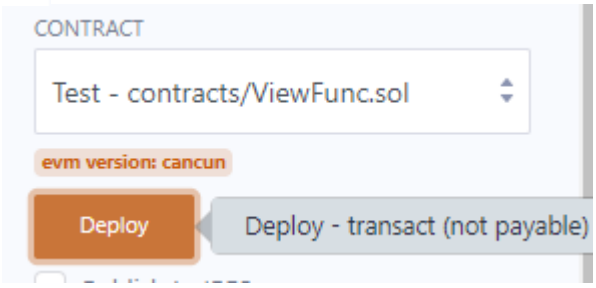
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Test{
    //Declaring state variable
    uint num1=2;
    uint num2=4;

    function getResult() public view returns (
        uint product,uint sum){
            product=num1*num2;
            sum=num1+num2;
        }
}
```

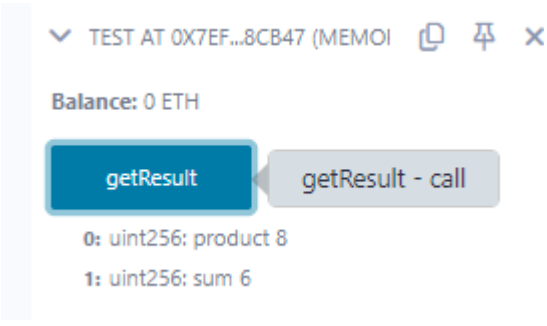
Code:-

pragma solidity 0.8.0;
contract Test{
uint num1=2;
uint num2=4;
function getResult() public view returns (uint product, uint sum){
product-num1*num2;
Sum=num1+num2;
}
}









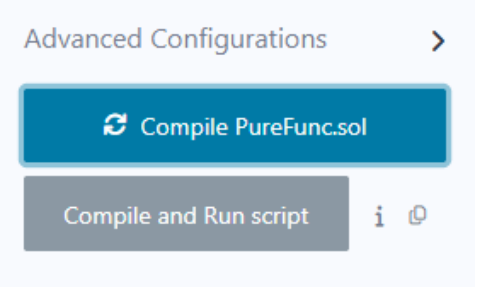
3] Pure Functions

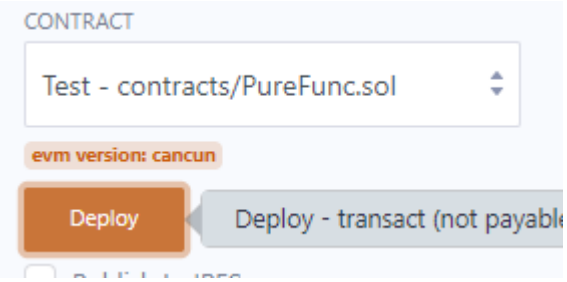
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Test{
    function getResult() public pure returns (uint product, uint sum){
        uint num1=2;
        uint num2=4;
        product=num1*num2;
        sum=num1+num1;
    }
}
```

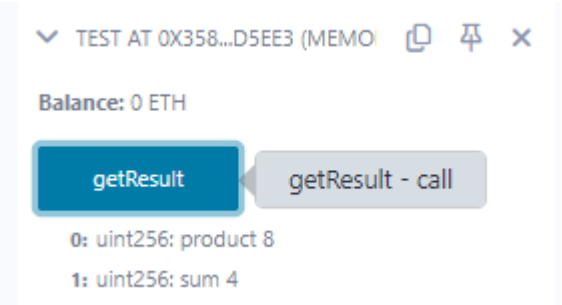
Code:-

```
pragma solidity ^0.8.0;
contract Test{
function getResult() public pure returns (uint product, uint sum){
uint num1-2;
uint num2=4;
product-num1*num2;
Sum=num1+num1;
}
}
```

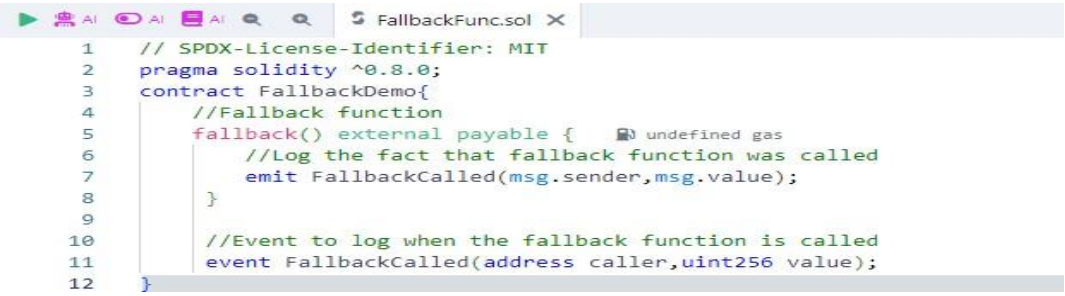
Output:







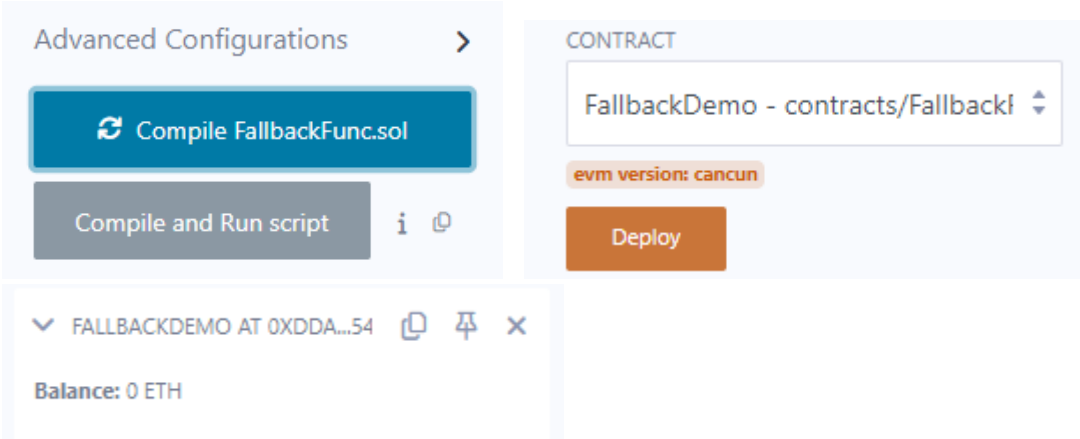
4] Fallback function



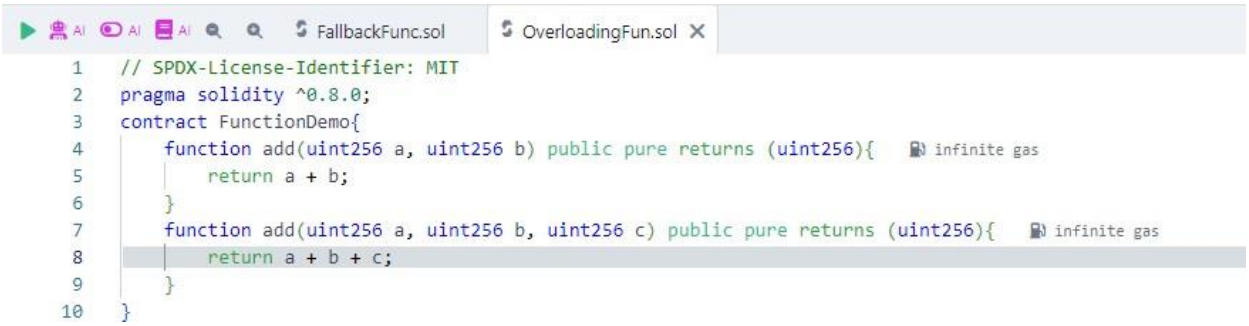
Code:-

```
pragma solidity 0.8.0;
contract FallbackDemo{
fallback() external payable {
emit FallbackCalled (msg.sender, msg.value);
}
event FallbackCalled (address caller, uint256 value);
}
```

Output:



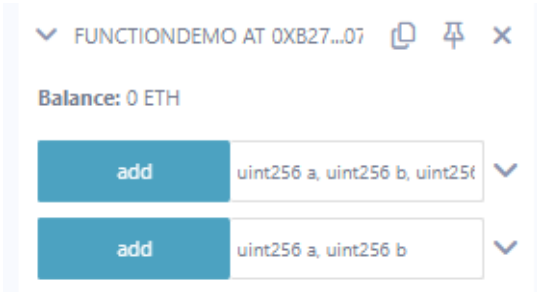
5] Function Overloading

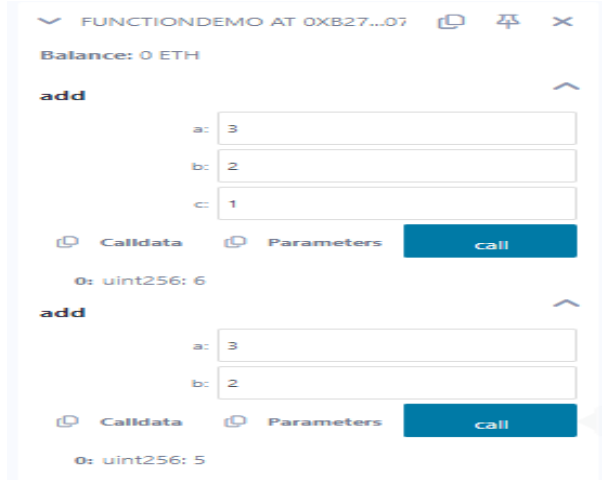


Code:-

```
pragma solidity 0.8.0;
contract FunctionDemo{
function add(uint256 a, uint256 b) public pure returns (uint256){
return a + b;
}
function add(uint256 a, uint256 b, uint256 c) public pure returns (uint256) {
 return a + b + c;
 }
 }
```

Output:



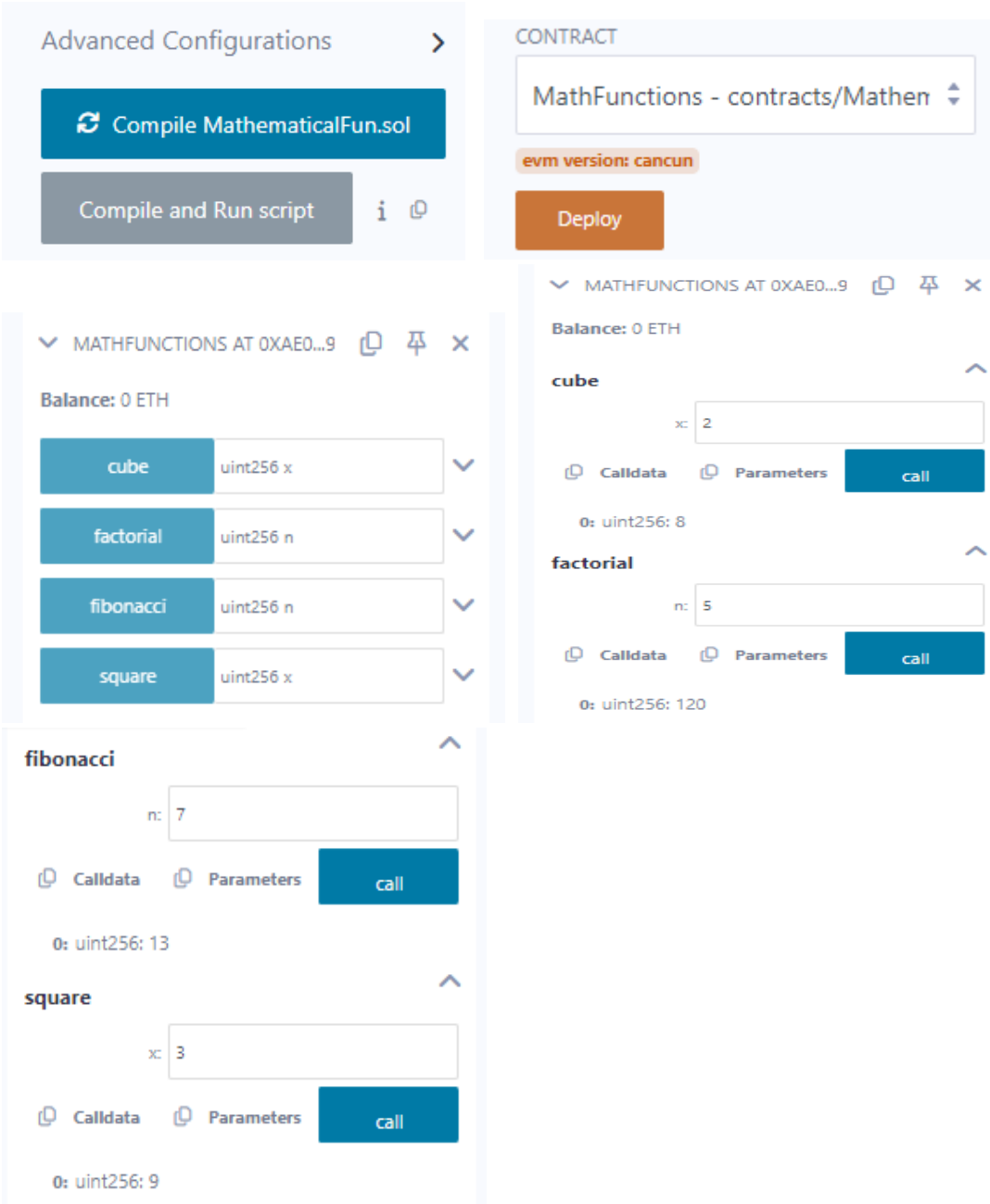


6] Mathematical functions

MathematicalFun.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MathFunctions{
    //Function to calculate the square of a number
    function square(uint256 x) public pure returns (uint256){    infinite gas
        return x * x;
    }

    //Function to calculate the cube of a number
    function cube(uint256 x) public pure returns (uint256){    infinite gas
        return x * x * x;
    }

    //Function to calculate the factorial of a number
    function factorial(uint256 n) public pure returns(uint256){    infinite gas
        uint256 result = 1;
        for (uint256 i =2;i<=n;i++){
            result *=i;
        }
        return result;
    }

    //Function to calculate the nth fibonacci number
    function fibonacci(uint256 n) public pure returns(uint256){    infinite gas
        if(n==0) return 0;
        uint256 a = 0;
        uint256 b = 1;
        for(uint256 i = 2; i<=n;i++){
            (a,b)=(b,a+b);
        }
        return b;
    }
}
```

Output:


Advanced Configurations
Compile MathematicalFun.sol
Compile and Run script
CONTRACT
MathFunctions - contracts/Mathen
evm version: cancun
Deploy
MATHFUNCTIONS AT 0XAE0...9
Balance: 0 ETH
cube
uint256 x
factorial
uint256 n
fibonacci
uint256 n
square
uint256 x
fibonacci
n: 7
Calldata
Parameters
call
0: uint256: 13
square
x: 3
Calldata
Parameters
call
0: uint256: 9
MATHFUNCTIONS AT 0XAE0...9
Balance: 0 ETH
cube
x: 2
Calldata
Parameters
call
0: uint256: 8
factorial
n: 5
Calldata
Parameters
call
0: uint256: 120

7] Cryptographic functions

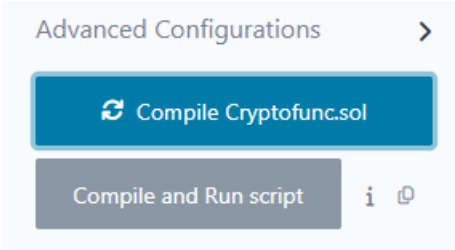
MathematicalFun.sol Cryptofunc.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract CryptoFunctions{
    //Function to hash data using keccal256
    function hashData(string memory _data) public pure returns (bytes32){    infinite gas
        return keccak256(abi.encodePacked(_data));
    }

    //Function to generate a random number using blockhash
    function random() public view returns (uint256){    infinite gas
        return uint256(blockhash(block.number-1))%100;
    }
}
```

Code:-

```
pragma solidity "0.8.0;
contract CryptoFunctions{
function hashData (string memory _data) public pure returns (bytes32){
return keccak256 (abi.encodePacked(_data));
}
function random() public view returns (uint256) {
return uint256(blockhash (block.number-1))%100;

}
}
```

Output:



Practical No: 06

Aim: Implement and demonstrate the use of the following in Solidity

1] Contracts

2] Inheritance

3] Abstract Class

1] Contracts

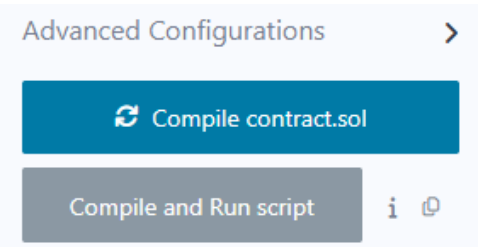
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

//base contract
contract Shape{
    uint internal area;

    //set area function
    function setArea(uint _area) external{    22520 gas
        area = _area;
    }

    //get area function
    function getArea() external view returns(uint){    2415 gas
        return area;
    }
}
//derived contract
contract Circle is Shape{
    uint internal radius;

    //constructor to set the radius
    constructor(uint _radius){    infinite gas 101200 gas
        radius = _radius;
    }

    //function to calculate the area of a circle
    function calculateArea() external{    infinite gas
        area = 3* radius * radius; //Approximation of area for simplicity
    }
}
```

Code:-

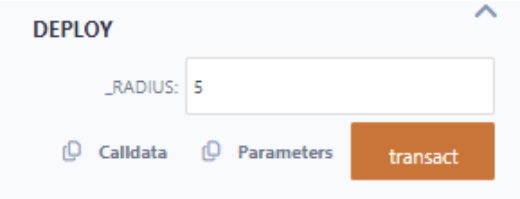
```
pragma solidity ^0.8.2;
//base contract contract Shape{
uint internal area;
//set area function
function setArea (uint area) external{
area = _area;
```

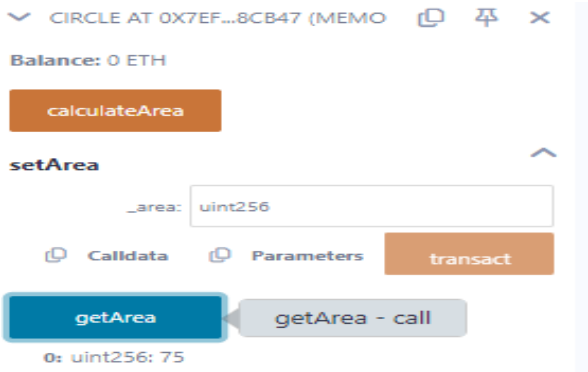
```
}
22520 gas
//get area function
function getArea() external view returns (uint) {
return area;
}
}
2415 gas
//derived contract
contract Circle is Shape
uint internal radius;
//constructor to set the radius
constructor (uint _radius) { infinite gas 101200 gas radius_radius;
}
//function to calculate the area of a circle
function calculateArea() external{ infinite gas
}
area 3* radius radius; //Approximation of area for simplicity
```

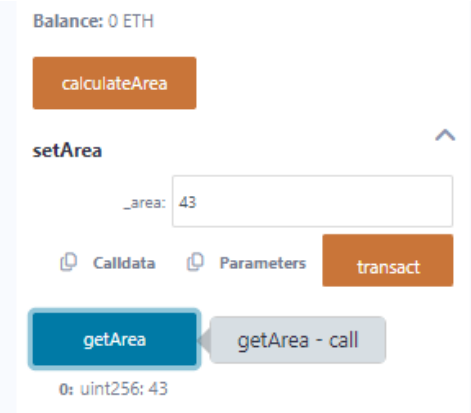
Output:


Advanced Configurations
Compile contract.sol
Compile and Run script


CONTRACT
Circle - contracts/contract.sol
evm version: cancun
Deploy
uint256 _radius


DEPLOY
_RADIUS: 5
Calldata
Parameters
transact


CIRCLE AT 0X7EF...8CB47 (MEMO
Balance: 0 ETH
calculateArea
setArea
_area: uint256
Calldata
Parameters
transact
getArea
getArea - call
0: uint256: 75


Balance: 0 ETH
calculateArea
setArea
_area: 43
Calldata
Parameters
transact
getArea
getArea - call
0: uint256: 43

2] Inheritance

Pract 7 inheritance.sol

```
//base contract
contract Person{
    string internal name;

    constructor(string memory _name){    infinite gas 110400 gas
        name = _name;
    }

    //function to get the person's name
    function getName() external view returns (string memory){    infinite gas
        return name;
    }

}

//derived contract for employee
contract Employee is Person{

    string internal position;

    constructor(string memory _name, string memory _position) Person(_name){    infinite gas 147800 gas
        position = _position;
    }

    //function to get the person's name
    function getPosition() external view returns (string memory){    infinite gas
        return position;
    }
}

//derived contract for student
contract Student is Person{

    uint internal grade;

    constructor(string memory _name, uint _grade) Person(_name){    infinite gas 131000 gas
        grade = _grade;
    }

    //function to get the students's grade
    function getGrade() external view returns (uint){    2437 gas
        return grade;
    }
}
```

Code:-

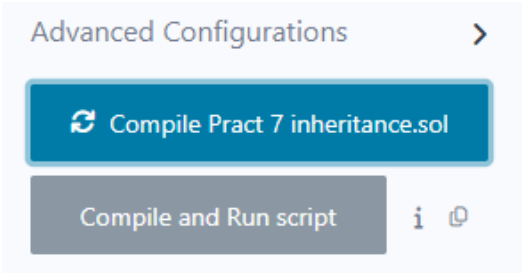
<

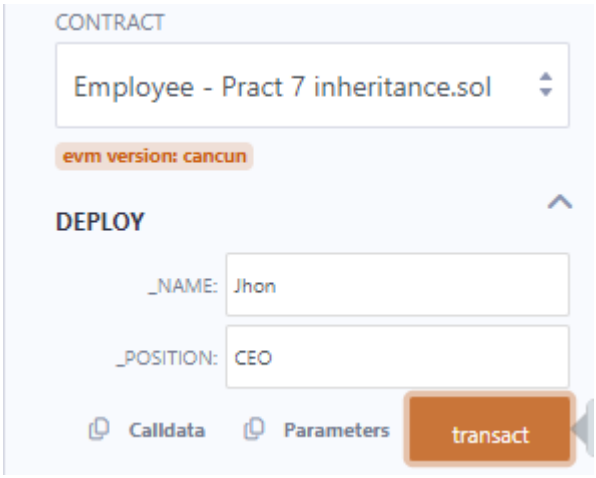
contract Person{

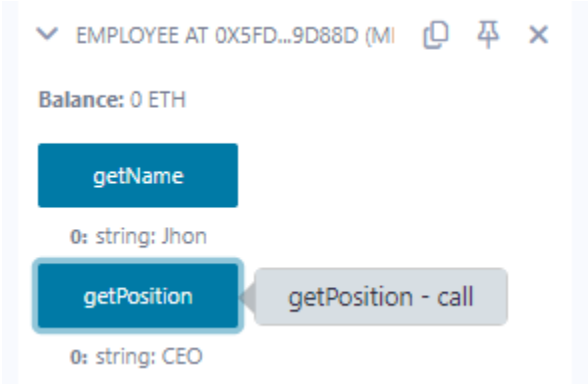
string internal name;

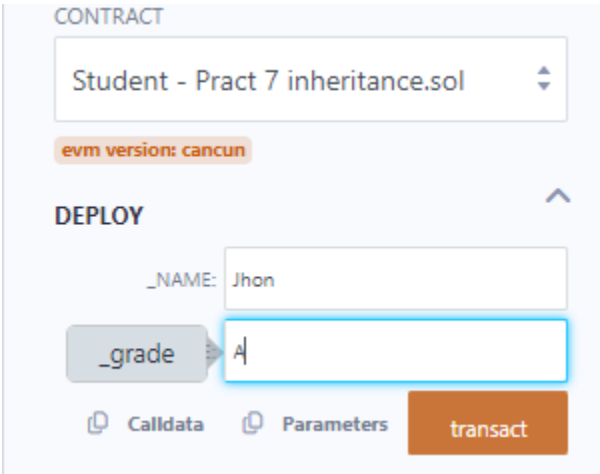
```
constructor(string memory _name) { name = _name;
infinite gas 110400 gas
}
//function to get the person's name
function getName() external view returns (string memory){
return name;
}
}
//derived contract for employee
contract Employee is Person{
string internal position;
constructor(string memory _name, string memory position) Person(_name){
 position position;

}
function getPosition() external view returns (string memory){
return position;
}
}
//derived contract for student
contract Student is Person{
uint internal grade;
constructor(string memory _name, uint grade) Person (_name){
grade = _grade;
}
//function to get the students's grade
function getGrade() external view returns (uint){
return grade;
}
}
```

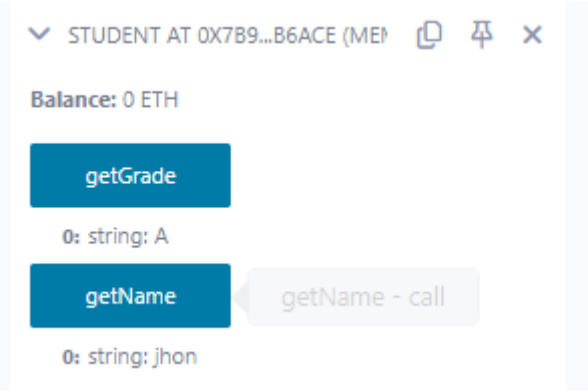
Output:











3] Abstract Class

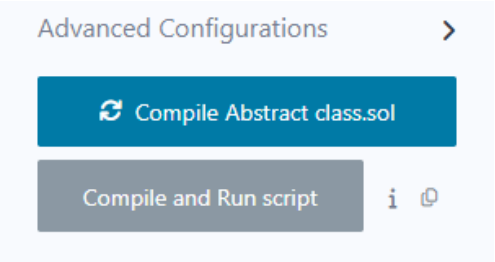
Abstract class.sol | Pract 7 inheritance.sol

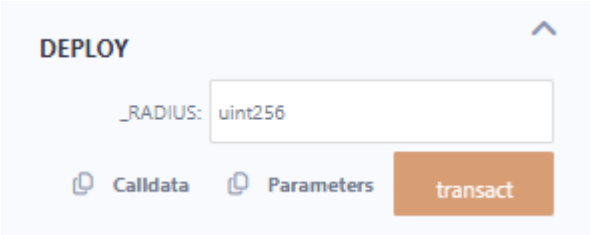
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

//abstract contract for shape
abstract contract Shape{
    //abstract contract to calculate area
    function calculateArea() external view virtual returns (uint);    - gas
}

//concrete contract for circle
contract Circle is Shape{

    uint internal radius;

    //constructor to set the radius
    constructor(uint _radius){    infinite gas 61400 gas
        radius = _radius;
    }

    //implementation of abstract function
    function calculateArea() external view override returns (uint){    infinite gas
        return 3 * radius * radius;//approximation of area for simplicity
    }
}
```

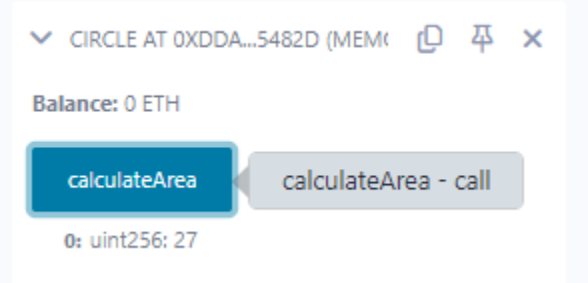
Code:-

```
pragma solidity ^0.8.0;
//abstract contract for shape.
abstract contract Shape{
function calculateArea() external view virtual returns (uint);

}
//concrete contract for circle
contract Circle is Shape{
        uint internal radius;
//constructor to set the radius
constructor (uint _radius) { infinite gas 61400 gas
radius = _radius;
}
//implementation of abstract function
function calculateArea() external view override returns (uint){
return 3* radius radius;//approximation of area for simplicity
}
}
```

Output:


Advanced Configurations
Compile Abstract class.sol
Compile and Run script


DEPLOY
_RADIUS:
uint256
Calldata
Parameters
transact


CIRCLE AT 0XDDA...5482D (MEM(
Balance: 0 ETH
calculateArea
calculateArea - call
0: uint256: 27

Practical No: 07

Aim: Implement and demonstrate the use of the following in Solidity.

1] Libraries

2] Events

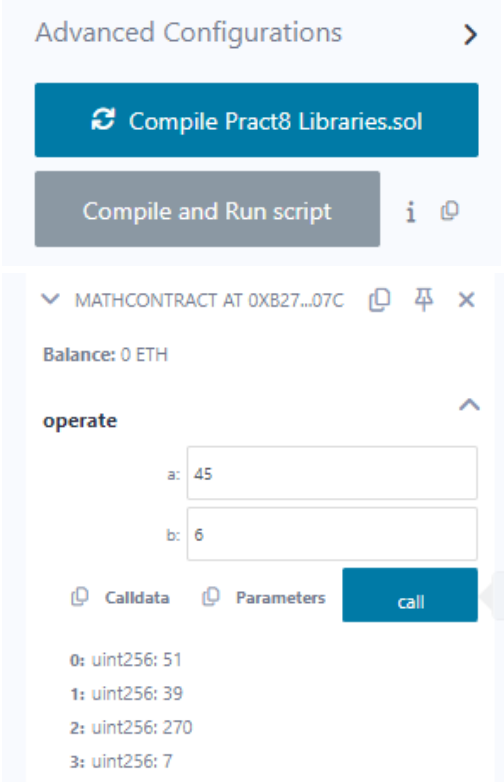
3] Error Handling

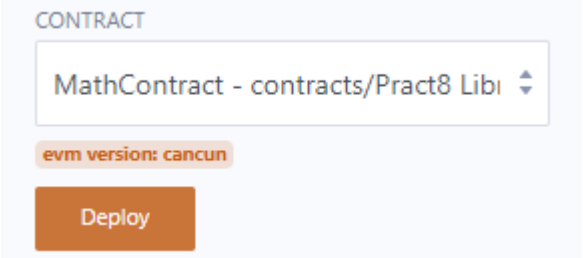
1] Libraries

Pract8 Libraries.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

library MathLib{
    //function to add two numbers
    function add(uint256 a , uint256 b) internal pure returns (uint256){    infinite gas
        return a+b;
    }

    //function to subtract two numbers
    function sub(uint256 a , uint256 b) internal pure returns (uint256){    infinite gas
        require(a>=b, "Subtraction result would be negative");
        return a-b;
    }

    //function to subtract two numbers
    function mul(uint256 a , uint256 b) internal pure returns (uint256){    infinite gas
        return a*b;
    }

    //function to subtract two numbers
    function div(uint256 a , uint256 b) internal pure returns (uint256){    infinite gas
        require(b != 0, "Division by zero");
        return a/b;
    }
}

contract MathContract{
    using MathLib for uint256;

    //fnction to perform arithematic operations using the library
    function operate(uint256 a, uint256 b) external pure returns (uint256, uint256, uint256, uint256){
        uint256 addition = a.add(b);
        uint256 subtraction = a.sub(b);
        uint256 multiplication = a.mul(b);
        uint division = a.div(b);
        return (addition,subtraction,multiplication,division);
    }
}
```

Output:





2] Events

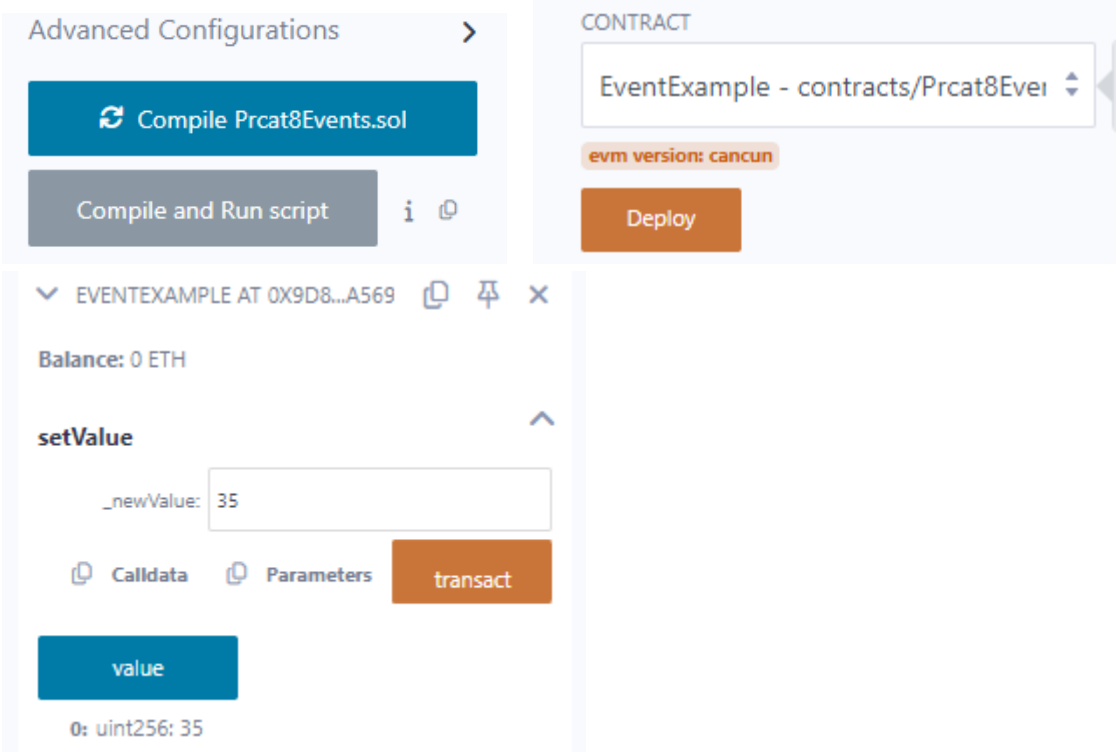
Prcat8Events.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract EventExample{
    event ValueSet(address indexed sender, uint newValue);

    uint public value;

    //function to set a new value and trigger an event
    function setValue(uint _newValue) external{    infinite gas
        value = _newValue;
        emit ValueSet(msg.sender, _newValue);
    }
}
```

Code:-

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract EventExample{
event ValueSet(address indexed sender, uint newValue);
uint public value;
//function to set a new value and trigger an event
function setValue(uint _newValue) external{
value = _newValue;
emit ValueSet (msg.sender, _newValue);
}
```

Output:



3] Error Handling

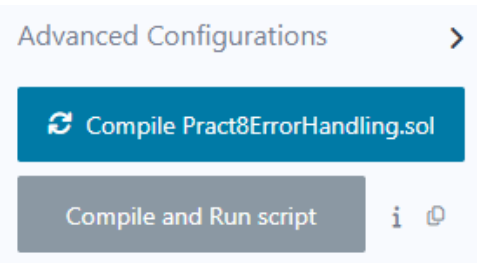
Pract 7 inheritance.sol | Pract8ErrorHandling.sol

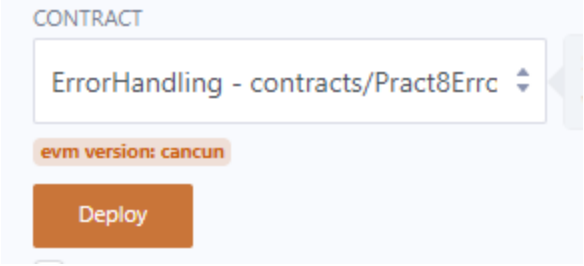
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract ErrorHandling {
mapping(address => uint256) public balances;
function withdraw(uint256 amount) public {    infinite gas
// Check if the sender has sufficient balance
require(amount <= balances[msg.sender], "Insufficient balance");
// If balance is sufficient, deduct the amount
balances[msg.sender] -= amount;
// Perform the withdrawal operation
// (In a real contract, this would involve transferring funds)
}
}
```

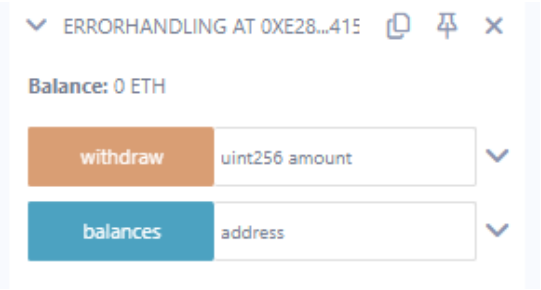
Code:-

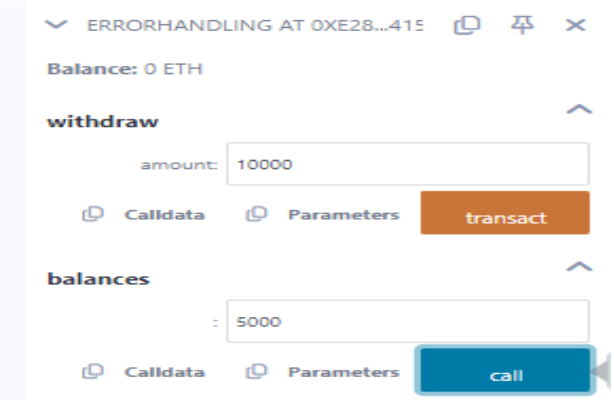
```
pragma solidity ^0.8.0;
contract ErrorHandling {
mapping (address => uint256) public balances;
 function withdraw(uint256 amount) public {
// Check if the sender has sufficient balance
infinite gas
require (amount <= balances [msg.sender], "Insufficient balance");
 // If balance is sufficient, deduct the amount
balances [msg.sender] -= amount;
// Perform the withdrawal operation
// (In a real contract, this would involve transferring funds)
}
}
```

Output:










```
transact to ErrorHandling.withdraw errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Insufficient balance".
You may want to cautiously increase the gas limit if the transaction went out of gas.

call to ErrorHandling.balances errored: Error encoding arguments: Error: invalid address (argument="address", value="5000", code=INVALID_ARGUMENT, version=ac
```