

**Name：Palak Gandhi**

**Class ：M.Sc IT - 2020-2021**

# INDEX

Vidyavardhini’s Vartak College of Arts, Commerce & Science Vasai Road(w) Palghar Dt 401202

M. Sc Information Technology Part II Semester IV 2020-2021

Block Chain Practical Index

|  |  |  |
| --- | --- | --- |
| Sr. No | Title | Page No |
| 1 | Practical 1a. A Simple Client class that generates private and public keys by  using built in Python RSA algorithm  Practical 1b  A transaction class to send and receive amount and use it  Practical 1c  Create multiple transactions and display them  Practical 1d  #Create a block chain genesis block and use it  Practical 1e  Create a mining function and test it | 3 |
| 2 | Install and configure Go Ethereum and Mist Browser |  |
| 3 | Implement and demonstrate the use of following in Solidity  a.Variable,operators,loops,DecisionMaking,String,Array,Enums,Structs,mapping,Coversion,Special variable.  b.Functions,Function Modifier, View Function, Pure function, call back function, function overloading, mathematical function, cryptographic function | 22 |
| 4 | Implement and demonstrate the use of following in Solidity  a. Withdrawal Pattern, Restricted access  b. Contracts, Inheritance, Constructors  c. Libraries, Assembly, Events, Error handling | 40 |
| 5 | Install Hyperledger fabric and composer. Deploy and execute application |  |
| 6 | Write a program to demonstrate mining of ether |  |
| 7 | Demonstrate the running of Blockchain node |  |
| 8 | Demonstrate the use of Bitcoin core API |  |
| 9 | Create your own blockchain and demonstrate its use |  |
| 10 | Build Daps with Angular |  |

# Practical 1

## Practical 1a

A Simple Client class that generates private and public keys by using built in Python RSA algorithm.

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Input:

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

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(self):

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

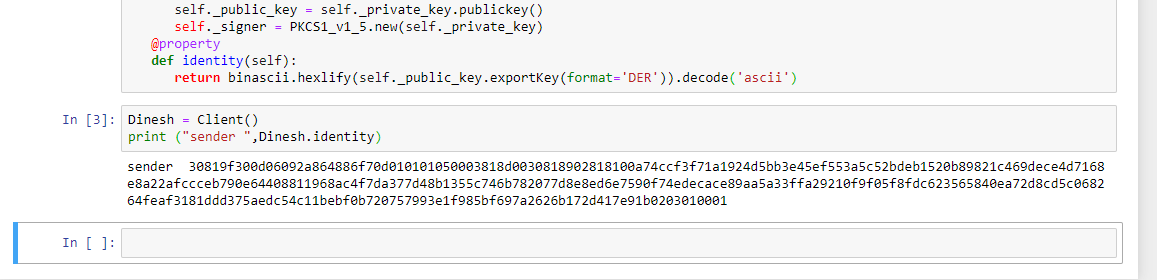
def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

Dinesh = Client()

print ("sender ",Dinesh.identity)

### Output:



## Practical 1b

A transaction class to send and receive amount and use it

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Input:

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

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(self):

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

class Transaction:

def \_\_init\_\_(self, sender, 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')

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()

t1 = Transaction(

Dinesh,

Ramesh.identity,

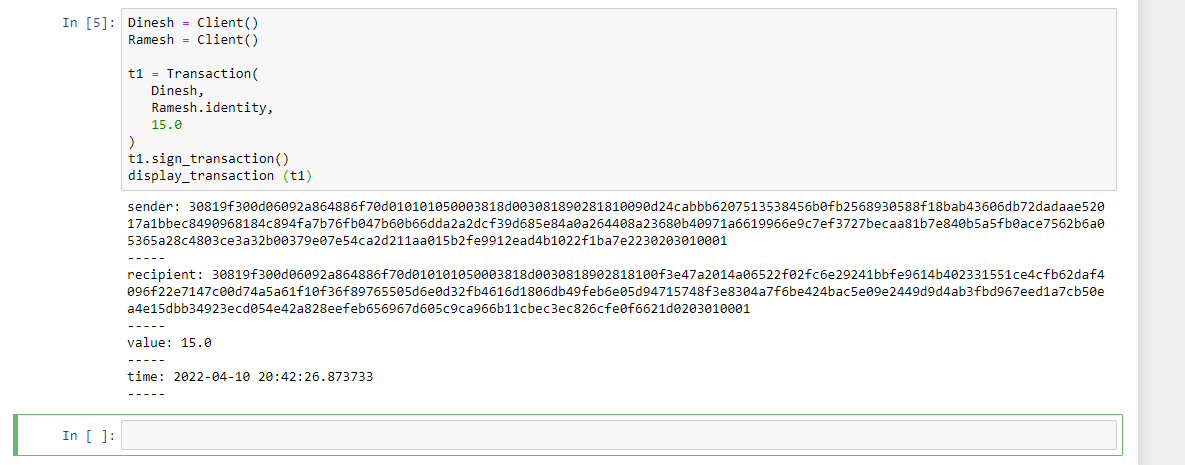
15.0

)

t1.sign\_transaction()

display\_transaction (t1)

### Output:



## Practical 1c

Create multiple transactions and display them

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Input:

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

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(self):

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

class Transaction:

def \_\_init\_\_(self, sender, 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')

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()

Suresh = Client()

t1 = Transaction(

Dinesh,

Ramesh.identity,

15.0

)

t1.sign\_transaction()

transactions.append(t1)

t2 = Transaction(

Ramesh,

Suresh.identity,

25.0

)

t2.sign\_transaction()

transactions.append(t2)

t3 = Transaction(

Ramesh,

Suresh.identity,

200.0

)

t3.sign\_transaction()

transactions.append(t3)

tn=1

for t in transactions:

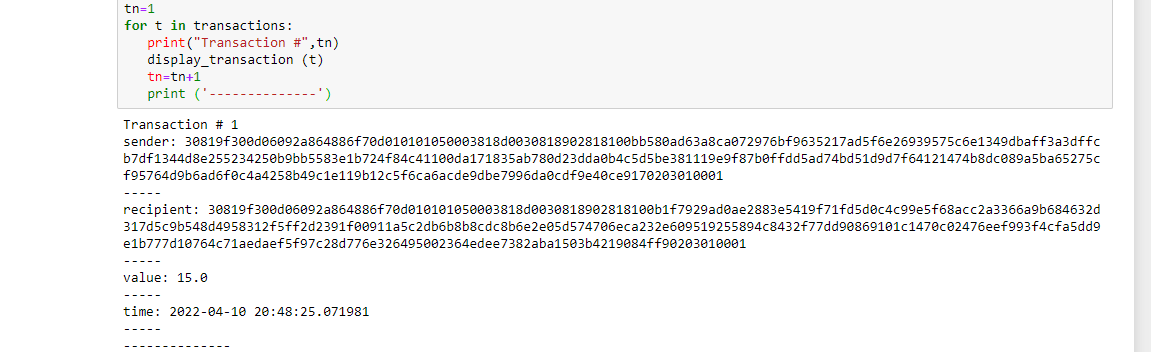
print("Transaction #",tn)

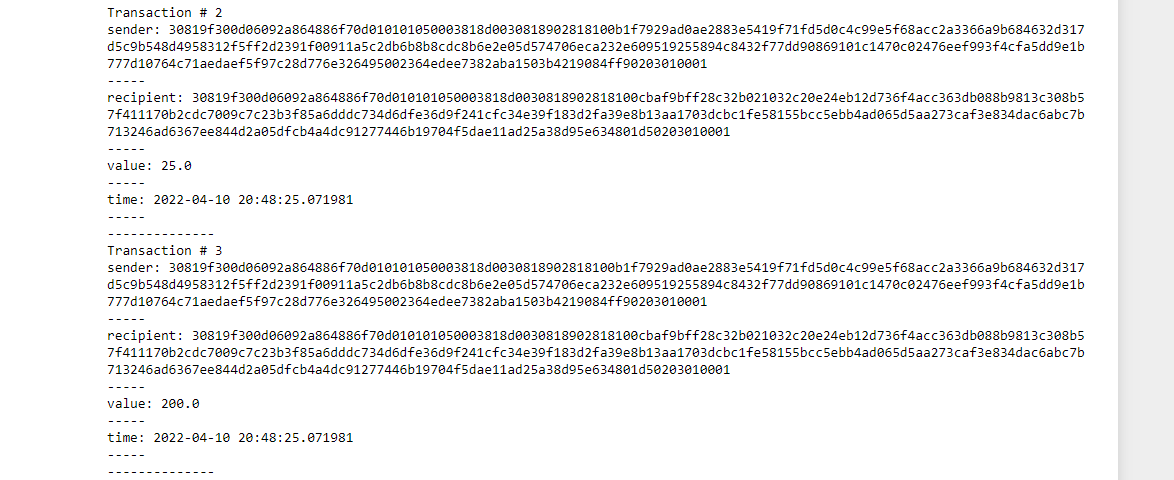
display\_transaction (t)

tn=tn+1

print ('--------------')

### Output:





## Practical 1d

Create a block chain genesis block and use it

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Input:

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

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(self):

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

class Transaction:

def \_\_init\_\_(self, sender, 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')

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 ('-----')

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 ('=====================================')

class Block:

def \_\_init\_\_(self):

self.verified\_transactions = []

self.previous\_block\_hash = ""

self.Nonce = ""

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)

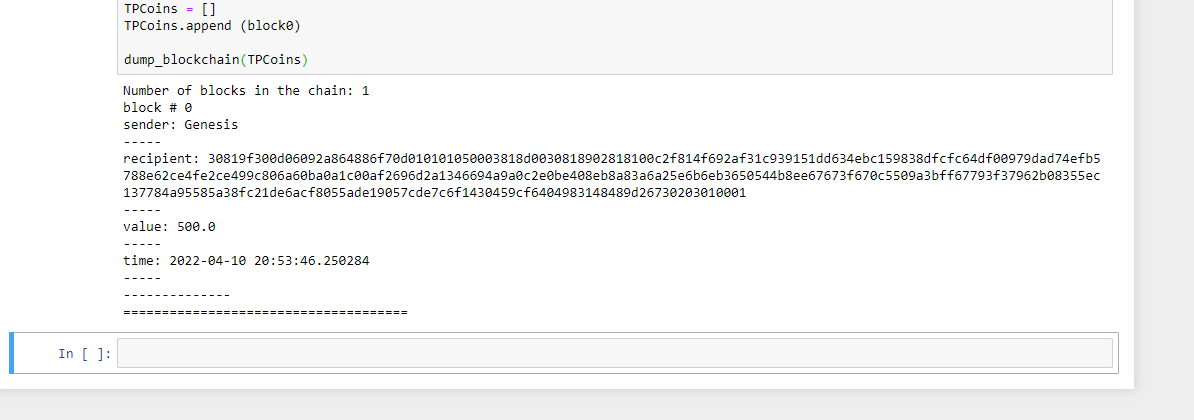
last\_block\_hash = digest

TPCoins = []

TPCoins.append (block0)

dump\_blockchain(TPCoins)

### Output:



## Practical 1e

Create a mining function and test it

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Input:

import hashlib

def sha256(message):

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

def mine(message, difficulty=1):

assert difficulty >= 1

#if(difficulty <1):

# return

#'1'\*3=> '111'

prefix = '1' \* difficulty

print("prefix",prefix)

for i in range(1000):

digest = sha256(str(hash(message)) + str(i))

print("testing=>"+digest)

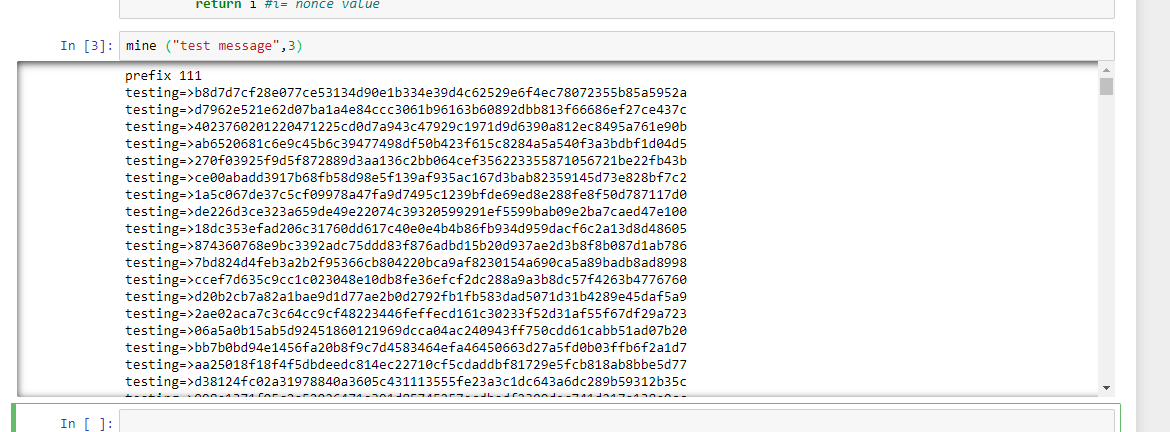
if digest.startswith(prefix):

print ("after " + str(i) + " iterations found nonce: "+ digest)

return i #i= nonce value

mine ("test message",3)

### Output:



## Practical 1f

add blocks to the miner and dump the blockchain

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Input:

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

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(self):

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

class Transaction:

def \_\_init\_\_(self, sender, 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')

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 ('-----')

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 ('=====================================')

class Block:

def \_\_init\_\_(self):

self.verified\_transactions = []

self.previous\_block\_hash = ""

self.Nonce = ""

def sha256(message):

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

def mine(message, difficulty=1):

assert difficulty >= 1

#if(difficulty <1):

# return

#'1'\*3=> '111'

prefix = '1' \* difficulty

for i in range(1000):

digest = sha256(str(hash(message)) + str(i))

if digest.startswith(prefix):

return i #i= nonce value

Dinesh = Client()

Ramesh =Client()

Vikas =Client()

t0 = Transaction (

"Genesis",

Dinesh.identity,

500.0

)

t1 = Transaction (

Ramesh,

Dinesh.identity,

40.0

)

t2 = Transaction (

Ramesh,

Dinesh.identity,

70.0

)

t3 = Transaction (

Vikas,

Ramesh.identity,

700.0

)

#blockchain

TPCoins = []

block0 = Block()

block0.previous\_block\_hash = None

Nonce = None

block0.verified\_transactions.append (t0)

digest = hash (block0)

last\_block\_hash = digest #last\_block\_hash it is hash of block0

TPCoins.append (block0)

block1 = Block()

block1.previous\_block\_hash = last\_block\_hash

block1.verified\_transactions.append (t1)

block1.verified\_transactions.append (t2)

block1.Nonce=mine (block1, 2)

digest = hash (block1)

last\_block\_hash = digest

TPCoins.append (block1)

block2 = Block()

block2.previous\_block\_hash = last\_block\_hash

block2.verified\_transactions.append (t3)

Nonce = mine (block2, 2)

block2.Nonce=mine (block2, 2)

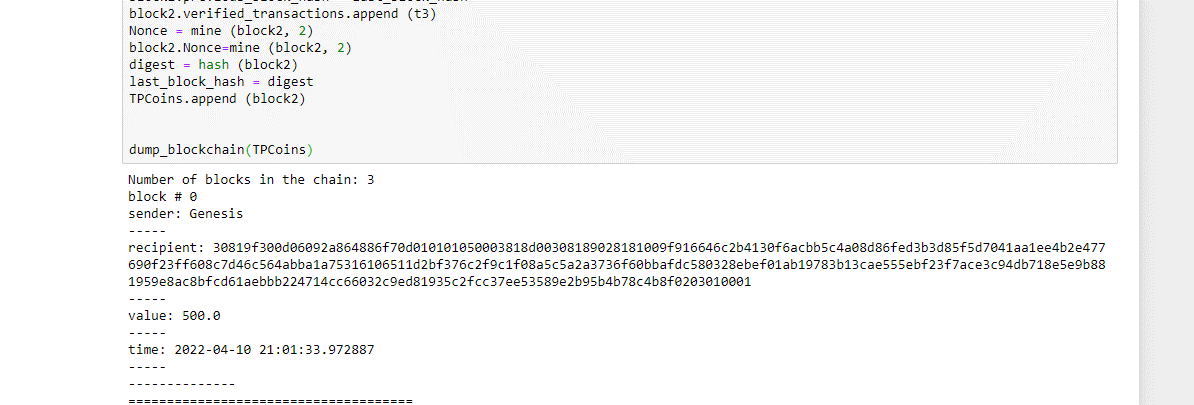
digest = hash (block2)

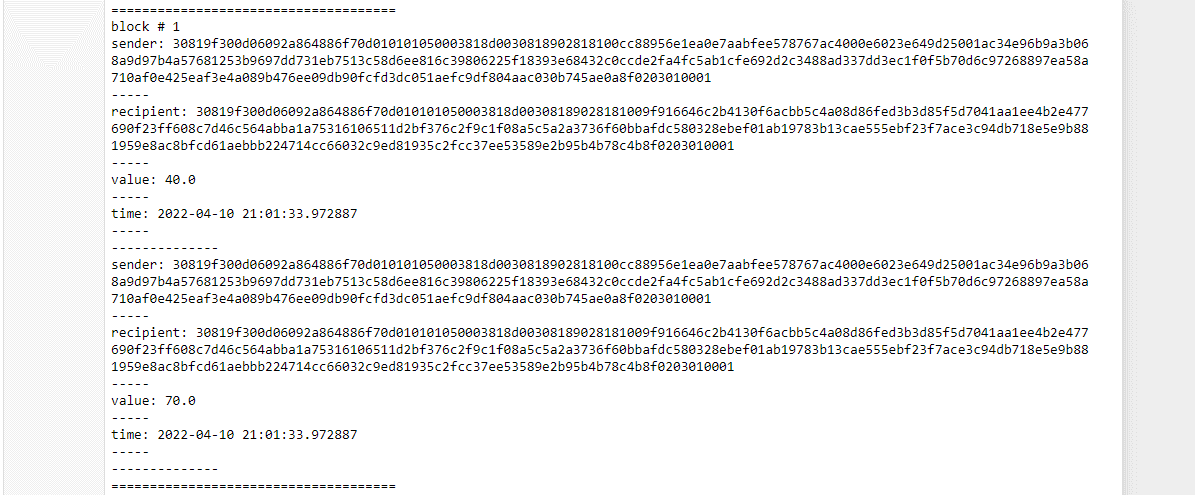
last\_block\_hash = digest

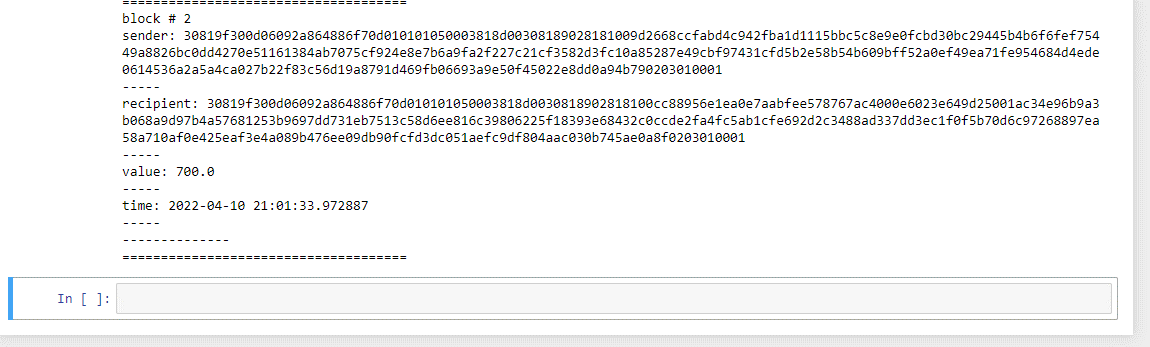
TPCoins.append (block2)

dump\_blockchain(TPCoins)

### Output:







# Practical 3

## Practical 3a

Variable, operators, loops, Decision Making, String, Array, Enums, Structs, mapping, Conversion, Special variable.

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Variable

#### Input:

// Solidity program to

// demonstrate how to

// use of variables

//SPDX-License-Identifier: GPL-3.0

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(

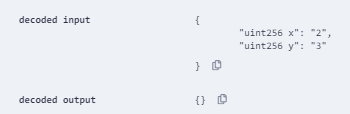
    ) public view returns (uint) {

        return sum;

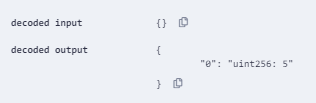
    }

}

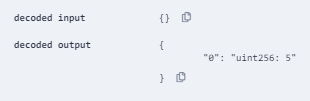
#### Output:



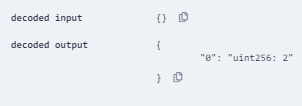
call to Test.get



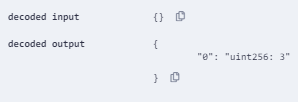
call to Test.sum



call to Test.var1



call to Test.var2



### Loop

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

// Solidity program to

// demonstrate how to

// write a smart contract

//SPDX-License-Identifier: GPL-3.0

pragma solidity >= 0.4.16 < 0.7.0;

contract Factorial {

    uint n;

    uint result=1;

    function  setn(uint a)  public  {

        n=a;

        uint i;

        for ( i=1;i<=n;i++)

        {

            result=result\*i;

        }

    }

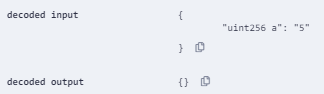
    function get() public view returns(uint) {

        return result;

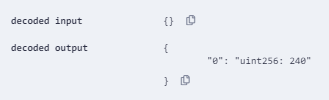
    }

}

#### Output:



call to Factorial.get



### Multi level inheritance

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

//Solidity program to

// demonstrate Multi-Level

// Inheritance

pragma solidity >=0.4.22 <0.6.0;

// Defining parent contract A

contract A {

    // Declaring state variables

    string internal x;

    string a = "Geeks" ;

    string b = "For";

    // Defining external function

    // to return concatenated string

    function getA() external{

        x = string(abi.encodePacked(a, b));

    }

}

// Defining child contract B

// inheriting parent contract A

contract B is A {

    // Declaring state variables

    // of child contract B

    string public y;

    string c = "Geeks";

    // Defining external function to

    // return concatenated string

    function getB() external payable returns(

    string memory){

        y = string(abi.encodePacked(x, c));

    }

}

// Defining child contract C

// inheriting parent contract A

contract C is B {

    // Defining external function

    // returning concatenated string

    // generated in child contract B

    function getC() external view returns(

    string memory){

        return y;

    }

}

// Defining calling contract

contract caller {

    // Creating object of child C

    C cc = new C();

    // Defining public function to

    // return final concatenated string

    function testInheritance(

    )

    public returns (

    string memory) {

        cc.getA();

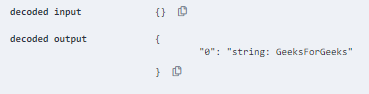
        cc.getB();

        return cc.getC();

    }

}

#### Output:



### Array

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

// demonstrate how to

// write a smart contract

//SPDX-License-Identifier: GPL-3.0

pragma solidity >= 0.4.16 < 0.7.0;

// Creating a contract

contract arraytest {

    // Defining the array

    uint[] data = [10, 20, 30, 40, 50];

    function array\_push() public  {

        data.push(60);

        data.push(70);

    }

    // Defining the function to push

    // values to the array

    function get (

    ) public  view returns(uint[] memory  ){

        return data;

    }

      function array\_pop(

    ) public   returns(uint[] memory){

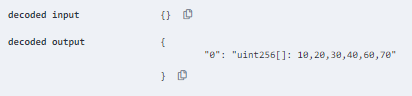
        data.pop();

        return data;

    }

}

#### Output:



### Enum

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

//Solidity program to demonstrate

// how to use 'enumerator'

//SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.5.0;

// Creating a contract

contract rainbowtest {

    // Creating an enumerator

    enum rainb

    {

      Violet,

      Indigo,

      Blue,

      Green,

      Yellow,

      Orange,

      Red

     }

    // Declaring variables of

    // type enumerator

    rainb r1;

    rainb choice;

    // Setting a default value

    rainb constant default\_value

      = rainb.Violet;

    // Defining a function to

    // set value of choice

    function set\_value() public {

      choice = rainb.Green;

    }

    // Defining a function to

    // return value of choice

    function get\_choice(

    ) public view returns (rainb) {

      return choice;

    }

    // Defining function to

    //  return default value

    function getdefaultvalue(

    ) public pure returns(rainb) {

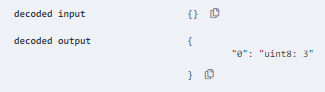
        return default\_value;

    }

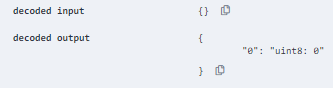
}

#### Output:

call to rainbowtest.get\_choice



call to rainbowtest.getdefaultvalue



### IF Else

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

//SPDX-License-Identifier: GPL-3.0

pragma solidity >0.5.0;

contract SolidityTest {

    constructor() public {

    }

    function getResult() public pure returns (uint) {

        uint a=100;

        uint b=40;

        uint result;

        if (a > b)

        {

            result=a-b;

        }

        else

        {

            result=b-a;

        }

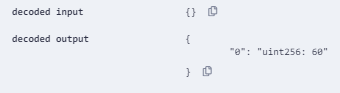
        return result;

    }

}

#### Output:

call to SolidityTest.getResult



## Practical 3b

Functions, Function Modifier, View Function, Pure function, call back function, function overloading, mathematical function, cryptographic function

### Cryptographic Function

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

contract Test {

   function callKeccak256() public pure returns(bytes32 result){

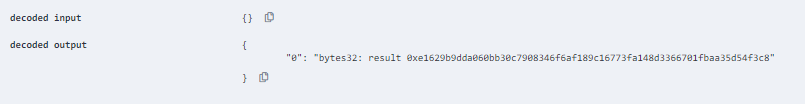
      return keccak256("ABC");

   }

}

#### Output:

call to Test.callKeccak256



### Fall Back Function

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

contract Test {

   uint public x ;

   function() external { x = 1; }

}

contract Sink {

   function() external payable { }

}

contract Caller {

   function callTest(Test test) public returns (bool) {

      (bool success,) = address(test).call(abi.encodeWithSignature("nonExistingFunction()"));

      require(success);

      // test.x is now 1

      address payable testPayable = address(uint160(address(test)));

      // Sending ether to Test contract,

      // the transfer will fail, i.e. this returns false here.

      return (testPayable.send(2 ether));

   }

   function callSink(Sink sink) public returns (bool) {

      address payable sinkPayable = address(sink);

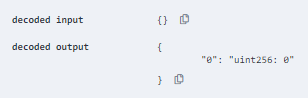
      return (sinkPayable.send(2 ether));

   }

}

#### Output:

call to Test.x



### Function Overloading

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

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){

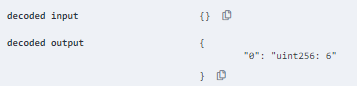
      return getSum(1,2,3);

   }

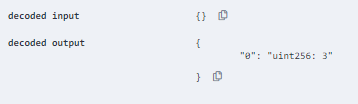
}

#### Output:

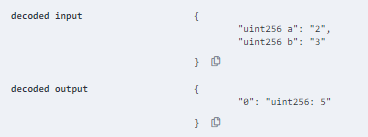
call to Test.callSumWithThreeArguments



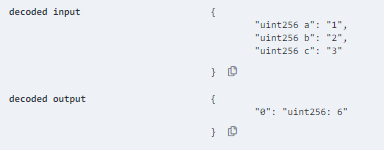
call to Test.callSumWithTwoArguments



call to Test.getSum



call to Test.getSum



### Mathematical Function

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

contract Test {

   function callAddMod() public pure returns(uint){

      return addmod(4, 5, 3);

   }

   function callMulMod() public pure returns(uint){

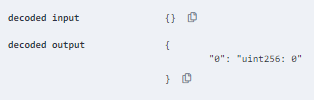
      return mulmod(4, 5, 3);

   }

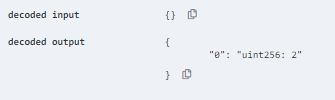
}

#### Output:

call to Test.callAddMod



call to Test.callMulMod



### Single Inheritance

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

// 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) {

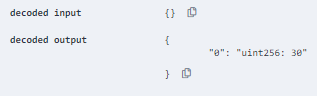
        return sum;

    }

}

#### Output:

call to child.getValue



# Practical 4

### Practical 4a

Implement and demonstrate the use of following in Solidity

#### Withdrawal Pattern

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

contract Test {

   address public richest;

   uint public mostSent;

   mapping (address => uint) pendingWithdrawals;

   constructor() public payable {

      richest = msg.sender;

      mostSent = msg.value;

   }

   function becomeRichest() public payable returns (bool) {

      if (msg.value > mostSent) {

         pendingWithdrawals[richest] += msg.value;

         richest = msg.sender;

         mostSent = msg.value;

         return true;

      } else {

         return false;

      }

   }

   function withdraw() public {

      uint amount = pendingWithdrawals[msg.sender];

      pendingWithdrawals[msg.sender] = 0;

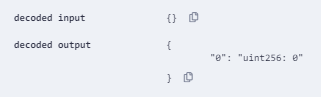
      msg.sender.transfer(amount);

   }

}

#### Output:

call to Test.mostSent



call to Test.richest



#### Restricted access

#### Input:

pragma solidity ^0.5.0;

contract Test {

   address public owner = msg.sender;

   uint public creationTime = now;

   modifier onlyBy(address \_account) {

      require(

         msg.sender == \_account,

         "Sender not authorized."

      );

      \_;

   }

   function changeOwner(address \_newOwner) public onlyBy(owner) {

      owner = \_newOwner;

   }

   modifier onlyAfter(uint \_time) {

      require(

         now >= \_time,

         "Function called too early."

      );

      \_;

   }

   function disown() public onlyBy(owner) onlyAfter(creationTime + 6 weeks) {

      delete owner;

   }

   modifier costs(uint \_amount) {

      require(

         msg.value >= \_amount,

         "Not enough Ether provided."

      );

      \_;

      if (msg.value > \_amount)

         msg.sender.transfer(msg.value - \_amount);

   }

   function forceOwnerChange(address \_newOwner) public payable costs(200 ether) {

      owner = \_newOwner;

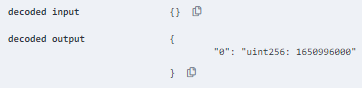
      if (uint(owner) & 0 == 1) return;

   }

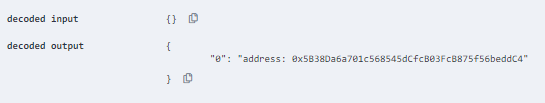
}

#### Output:

call to Test.creationTime



call to Test.owner



## Practical 4b

Implement and demonstrate the use of following in Solidity

#### Contracts

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

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; }

}

//External Contract

contract D {

   function readData() public returns(uint) {

      C c = new C();

      c.updateData(7);

      return c.getData();

   }

}

//Derived Contract

contract E is C {

   uint private result;

   C private c;

   constructor() public {

      c = new C();

   }

   function getComputedResult() public {

      result = compute(3, 5);

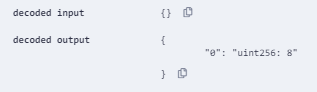
   }

   function getResult() public view returns(uint) { return result; }

   function getData() public view returns(uint) { return c.info(); }

}

#### Output:



#### Inheritance

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

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);

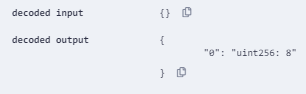
   }

   function getResult() public view returns(uint) { return result; }

   function getData() public view returns(uint) { return c.info(); }

}

#### Output:



#### Constructors

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

contract Base {

   uint data;

   constructor(uint \_data) public {

      data = \_data;

   }

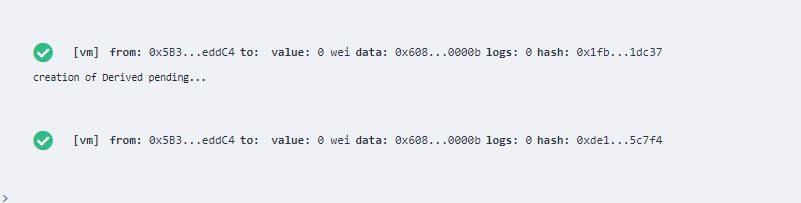
}

contract Derived is Base {

   constructor(uint \_info) Base(\_info \* \_info) public {}

}

#### Output:



## Practical 4c

#### Libraries

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

library Search {

   function indexOf(uint[] storage self, uint value) public view returns (uint) {

      for (uint i = 0; i < self.length; i++) if (self[i] == value) return i;

      return uint(-1);

   }

}

contract Test {

   uint[] data;

   constructor() public {

      data.push(1);

      data.push(2);

      data.push(3);

      data.push(4);

      data.push(5);

   }

   function isValuePresent() external view returns(uint){

      uint value = 4;

      //search if value is present in the array using Library function

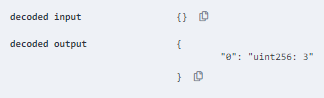
      uint index = Search.indexOf(data, value);

      return index;

   }

}

#### Output:



#### Assembly

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

library Sum {

   function sumUsingInlineAssembly(uint[] memory \_data) public pure returns (uint o\_sum) {

      for (uint i = 0; i < \_data.length; ++i) {

         assembly {

            o\_sum := add(o\_sum, mload(add(add(\_data, 0x20), mul(i, 0x20))))

         }

      }

   }

}

contract Test {

   uint[] data;

   constructor() public {

      data.push(1);

      data.push(2);

      data.push(3);

      data.push(4);

      data.push(5);

   }

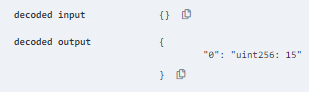
   function sum() external view returns(uint){

      return Sum.sumUsingInlineAssembly(data);

   }

}

#### Output:



#### Events

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

contract Test {

   event Deposit(address indexed \_from, bytes32 indexed \_id, uint \_value);

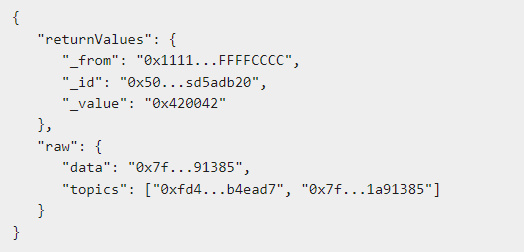
   function deposit(bytes32 \_id) public payable {

      emit Deposit(msg.sender, \_id, msg.value);

   }

}

#### Output:



#### Error handling

Summary::

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

#### Input:

pragma solidity ^0.5.0;

contract Vendor {

   address public seller;

   modifier onlySeller() {

      require(

         msg.sender == seller,

         "Only seller can call this."

      );

      \_;

   }

   function sell(uint amount) public payable onlySeller {

      if (amount > msg.value / 2 ether)

         revert("Not enough Ether provided.");

      // Perform the sell operation.

   }

}

#### Output:

