**Practical 1**

Aim: Creating a single transactions and displaying them

Codes

!pip install pycryptodome

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

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

  def \_\_init\_\_(self):

    random\_generator=Crypto.Random.new().read

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

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

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

  @property

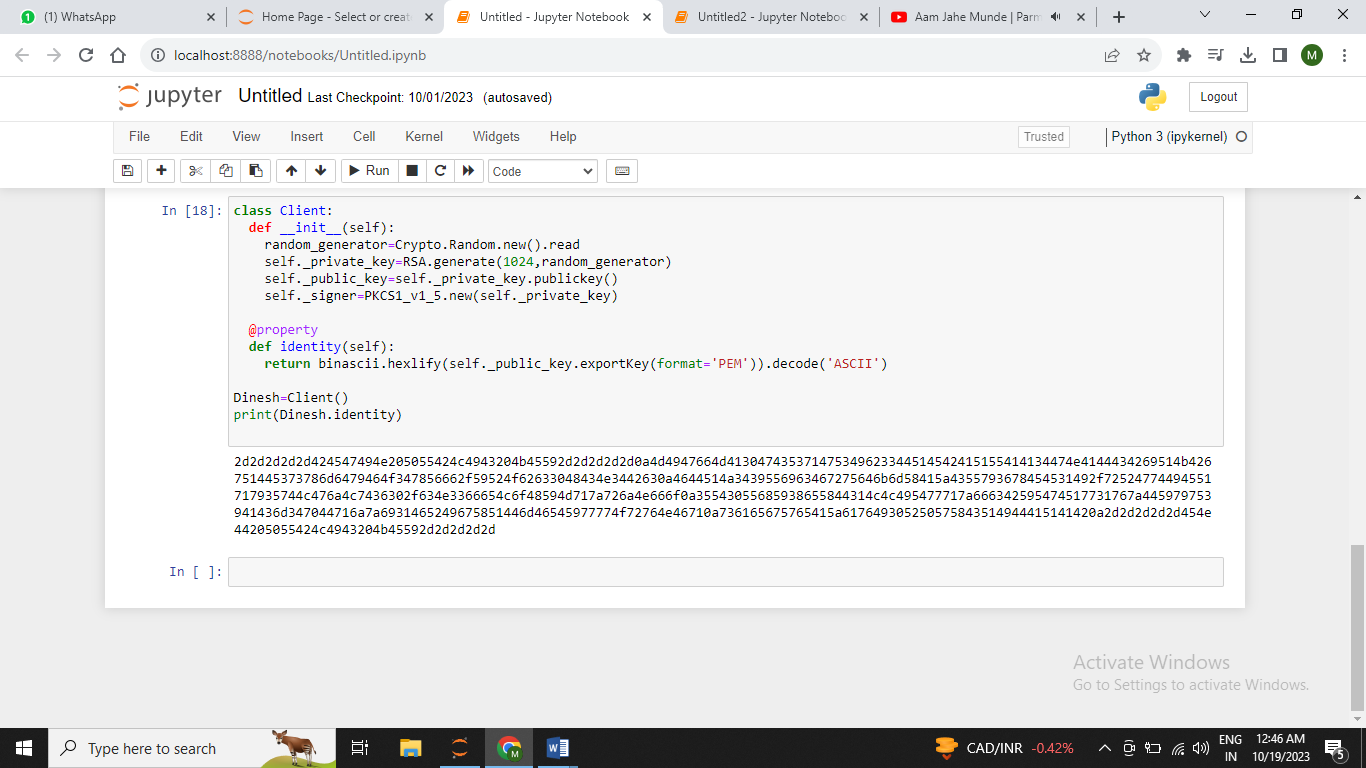
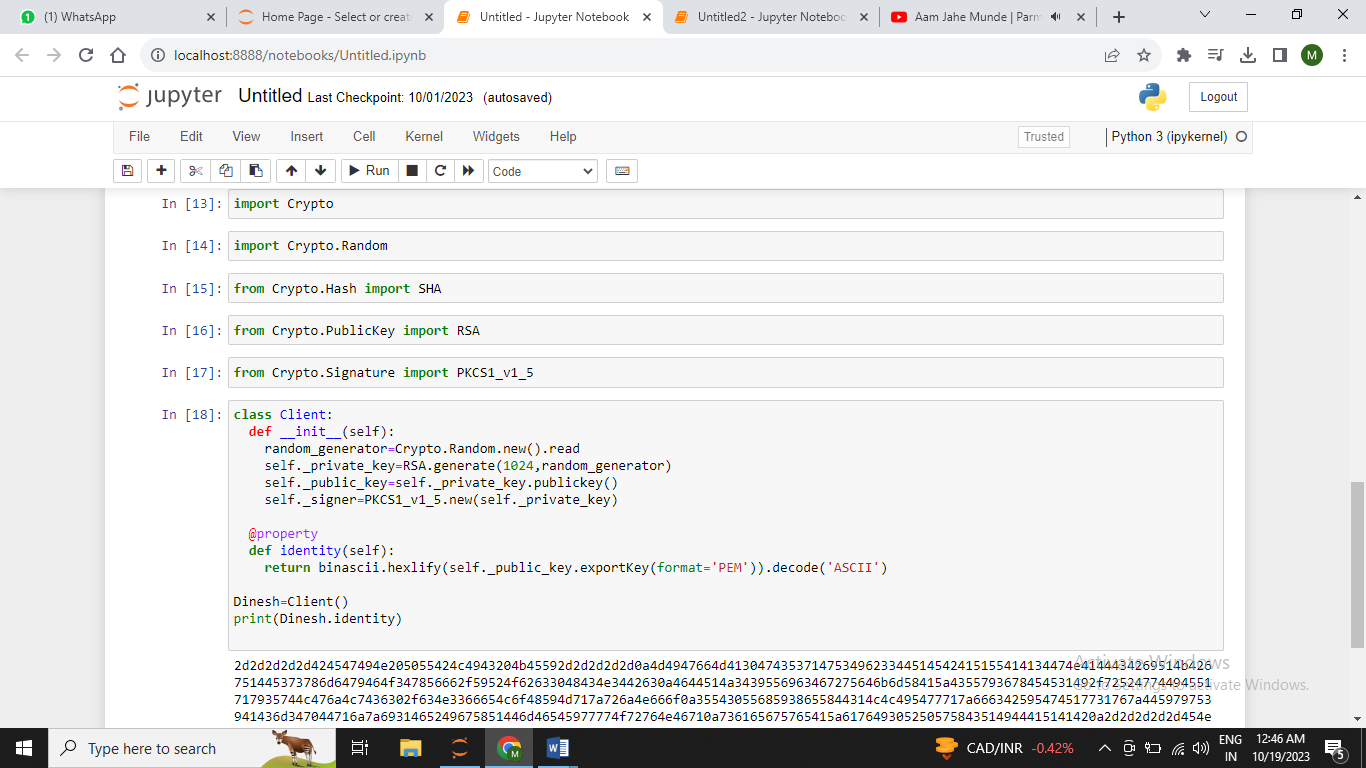
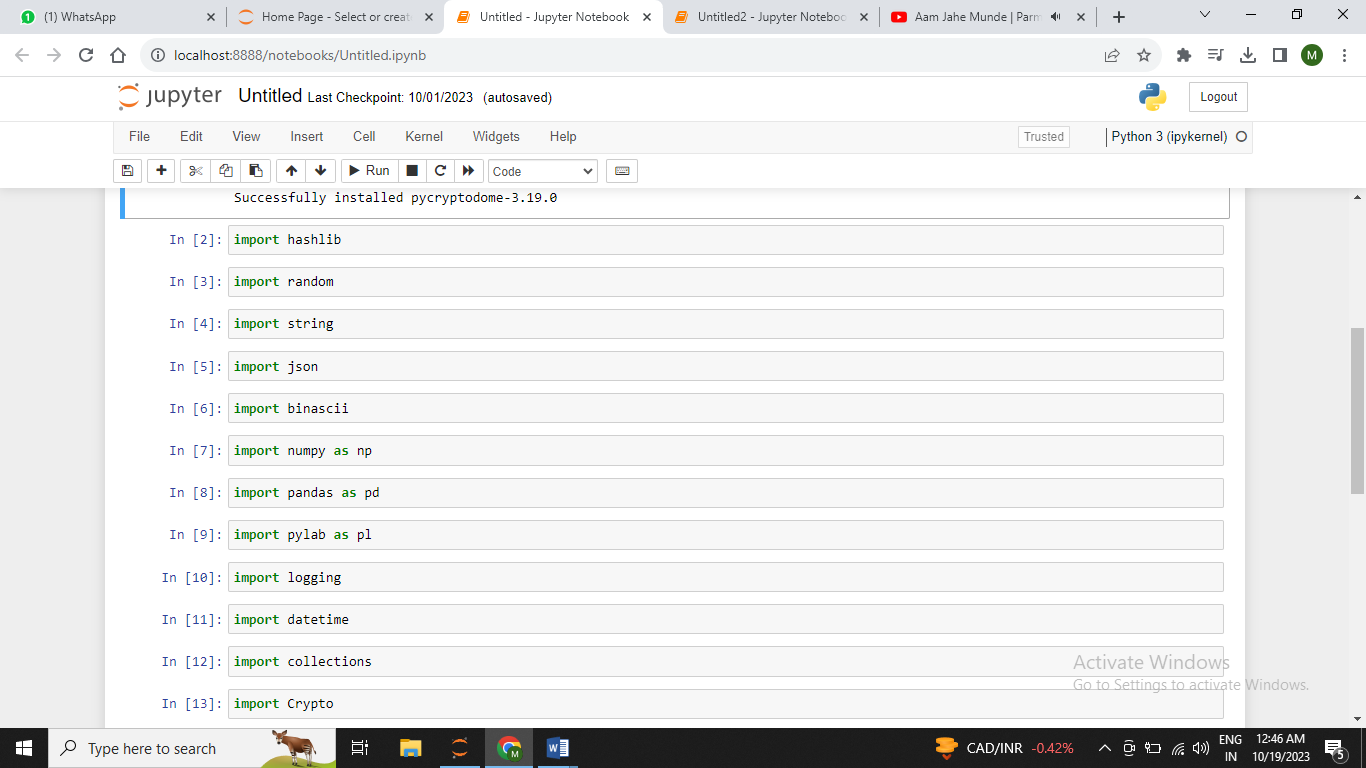
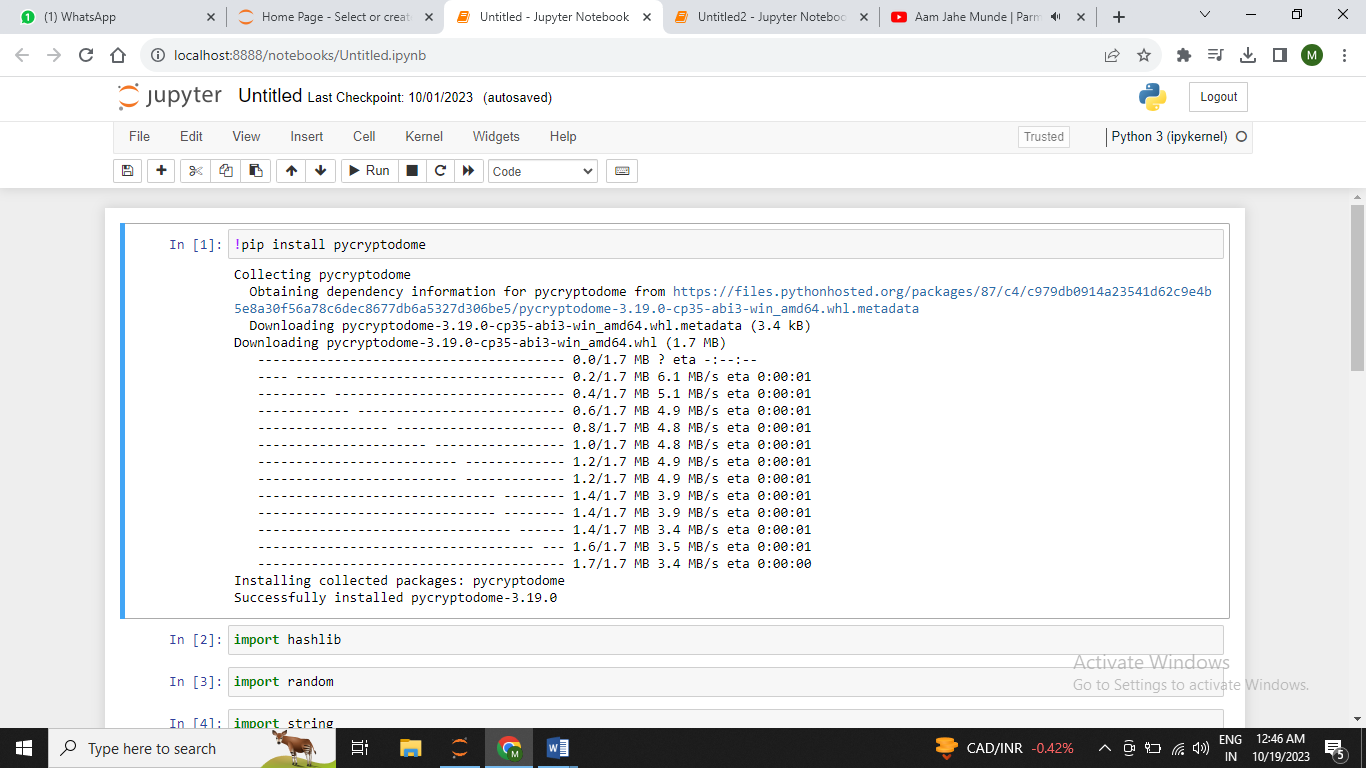
  def identity(self):

    return binascii.hexlify(self.\_public\_key.exportKey(format='PEM')).decode('ASCII')

Dinesh=Client()

print(Dinesh.identity)

Output:



**Practical 2**

Aim:A transaction class to receive and transact money and test it

Codes:

class Bank:

  def \_\_init\_\_(self):

    self.balance=0

    print("Chelsia\_283\n")

    print("The account is created")

  def deposit(self):

    amount=float(input("Enter the amount to be deposit:"))

    self.balance=self.balance+amount

    print("The deposit is succcessful and the balance in the account is %f" %self.balance)

  def withdraw(self):

    amount=float(input("Enter the amount to withdraw:"))

    if(self.balance>=amount):

      self.balance=self.balance-amount

      print("The withdraw is successful and the balance is %f" %self.balance)

    else:

      print('Insufficient Balance')

  def enquiry(self):

    print("Balance in the amount is %f" %self.balance)

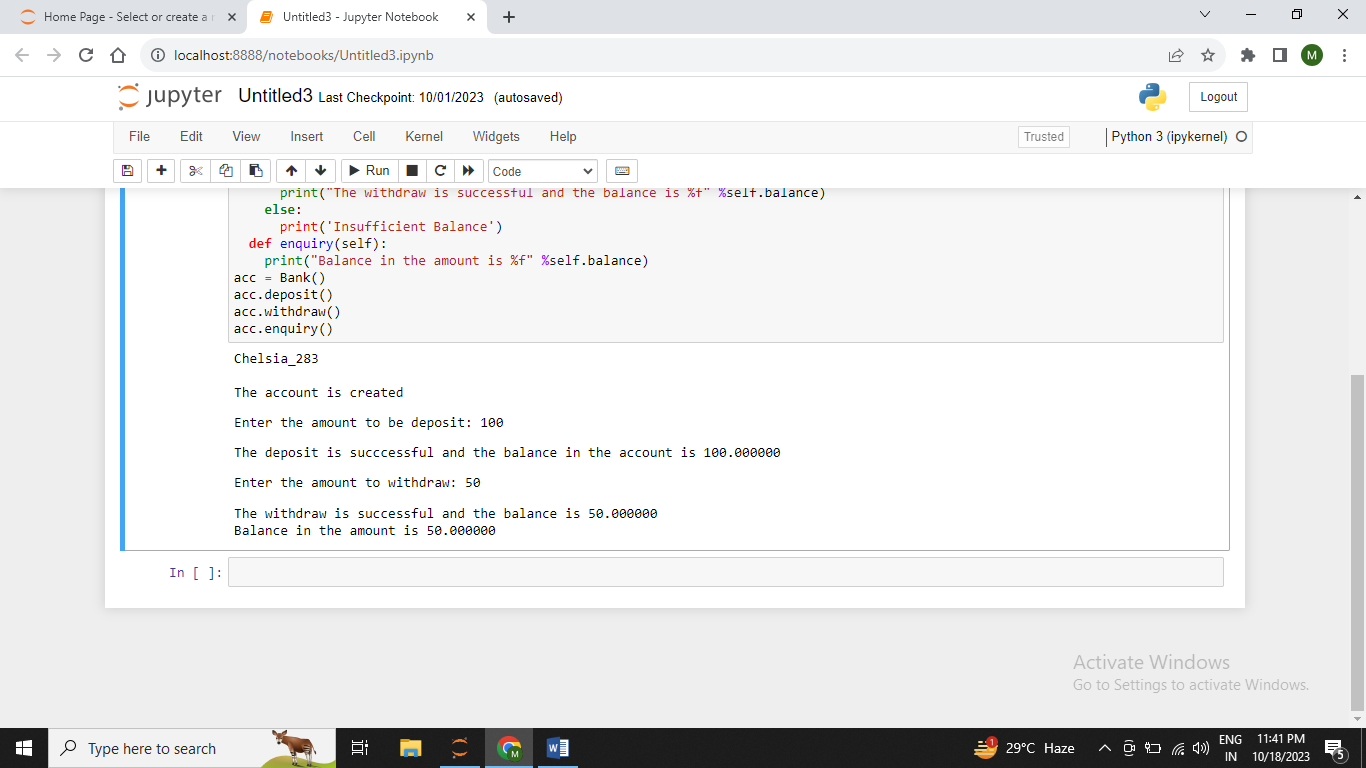
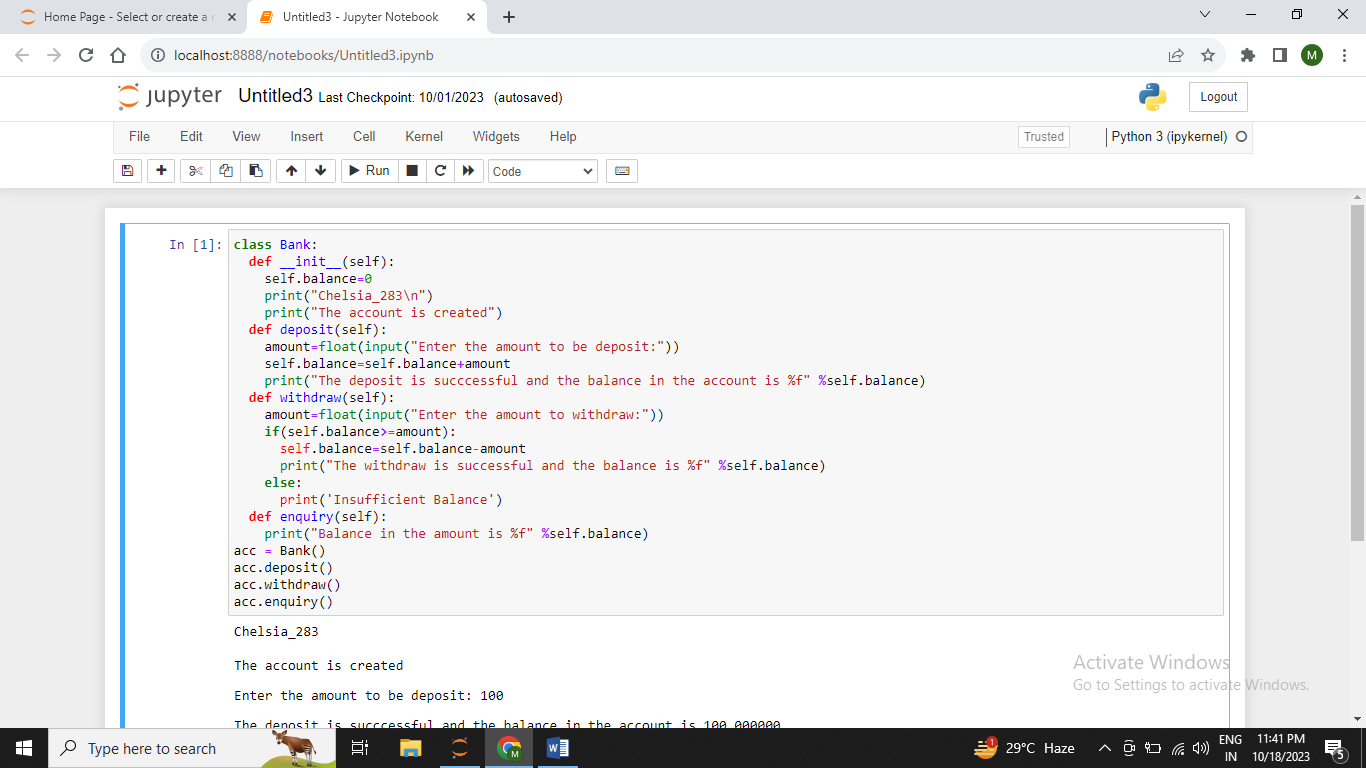
acc = Bank()

acc.deposit()

acc.withdraw()

acc.enquiry()

Output:



**Practical 3**

Aim:Create a Multiple Transaction and display them.

Codes:

!pip install pycryptodome

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

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto import Signature

class Client:

  def \_\_init\_\_(self):

    random\_generator=Crypto.Random.new().read

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

    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='PEM')).decode('ASCII')

Dinesh=Client()

print(Dinesh.identity)

class Transaction:

  def \_\_init\_\_(self,sender,recipient,value):

    self.sender=sender

    self.recipient=recipient

    self.value=value

    self.time=datetime.datetime.now()

  def to\_dict(self):

    if self.sender=="Genesis":

      identity="Genesis"

    else:

      identity=self.sender.identity

    return collections.OrderedDict({

        'sender':identity,

        'recipient':self.recipient,

        'value':self.value,

        'time':self.time})

  def sign\_transaction(self):

    private\_key=self.sender.\_private\_key

    signer=PKCS1\_v1\_5.new(private\_key)

    h=SHA.new(str(self.to\_dict()).encode('utf8'))

    return binascii.hexlify(signer.sign(h)).decode('ascii')

Dinesh=Client()

Ramesh=Client()

t=Transaction(

    Dinesh,

    Ramesh.identity,

    5.0

)

signature=t.sign\_transaction()

print(Signature)

print("=============Multiple Transaction=============")

def display\_transaction(transaction):

  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('------')

Dinesh=Client()

Ramesh=Client()

Seema=Client()

Vijay=Client()

transactions=[]

t1=Transaction(

    Dinesh,

    Ramesh.identity,

    15.0

)

t1.sign\_transaction()

transactions.append(t1)

t2=Transaction(

    Dinesh,

    Seema.identity,

    6.0

)

t2.sign\_transaction()

transactions.append(t2)

t3=Transaction(

    Ramesh,

    Vijay.identity,

    2.0

)

t3.sign\_transaction()

transactions.append(t3)

t4=Transaction(

    Seema,

    Ramesh.identity,

    4.0

)

t4.sign\_transaction()

transactions.append(t4)

t5=Transaction(

    Vijay,

    Seema.identity,

    7.0

)

t5.sign\_transaction()

transactions.append(t5)

t6=Transaction(

    Ramesh,

    Seema.identity,

    3.0

)

t6.sign\_transaction()

transactions.append(t6)

t7=Transaction(

    Seema,

    Dinesh.identity,

    8.0

)

t7.sign\_transaction()

transactions.append(t7)

t8=Transaction(

    Seema,

    Ramesh.identity,

    1.0

)

t8.sign\_transaction()

transactions.append(t8)

t9=Transaction(

    Vijay,

    Dinesh.identity,

    5.0

)

t9.sign\_transaction()

transactions.append(t9)

t10=Transaction(

    Vijay,

    Ramesh.identity,

    3.0

)

t10.sign\_transaction()

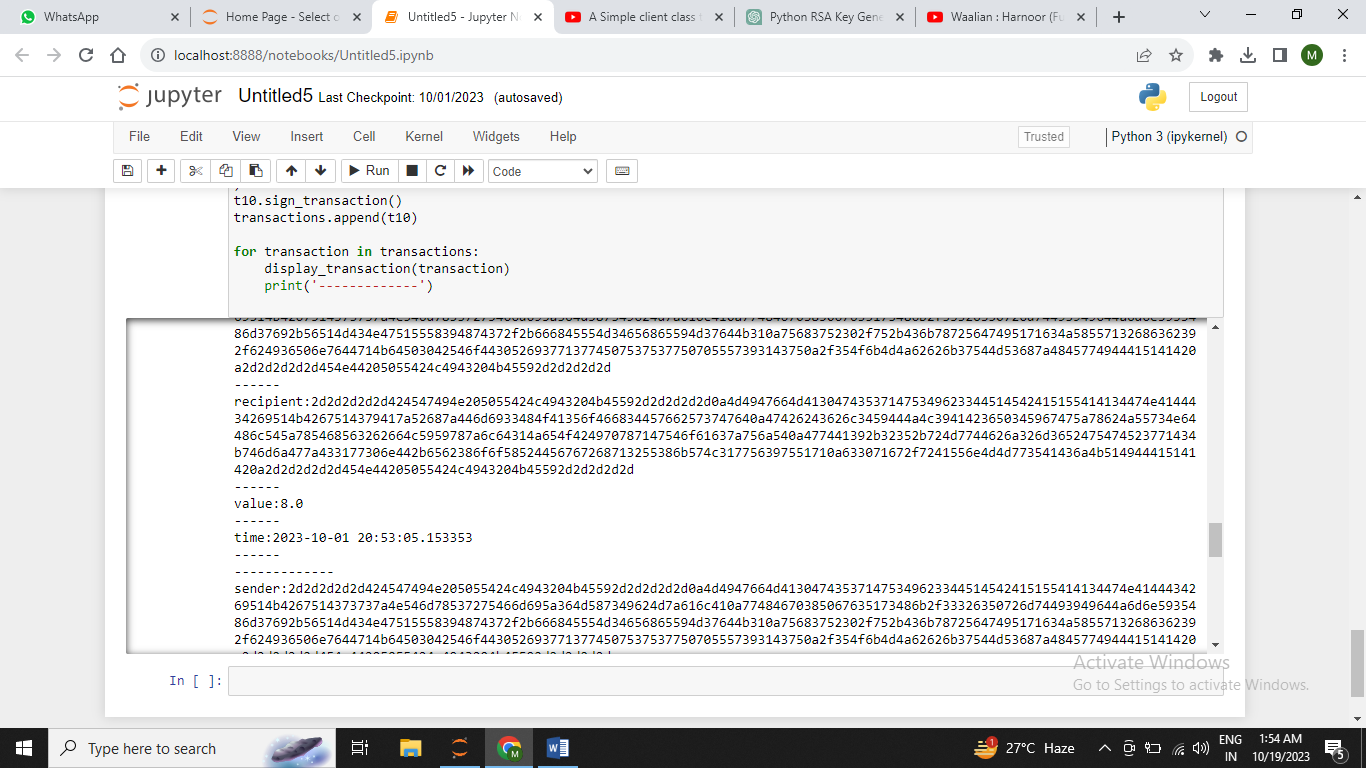
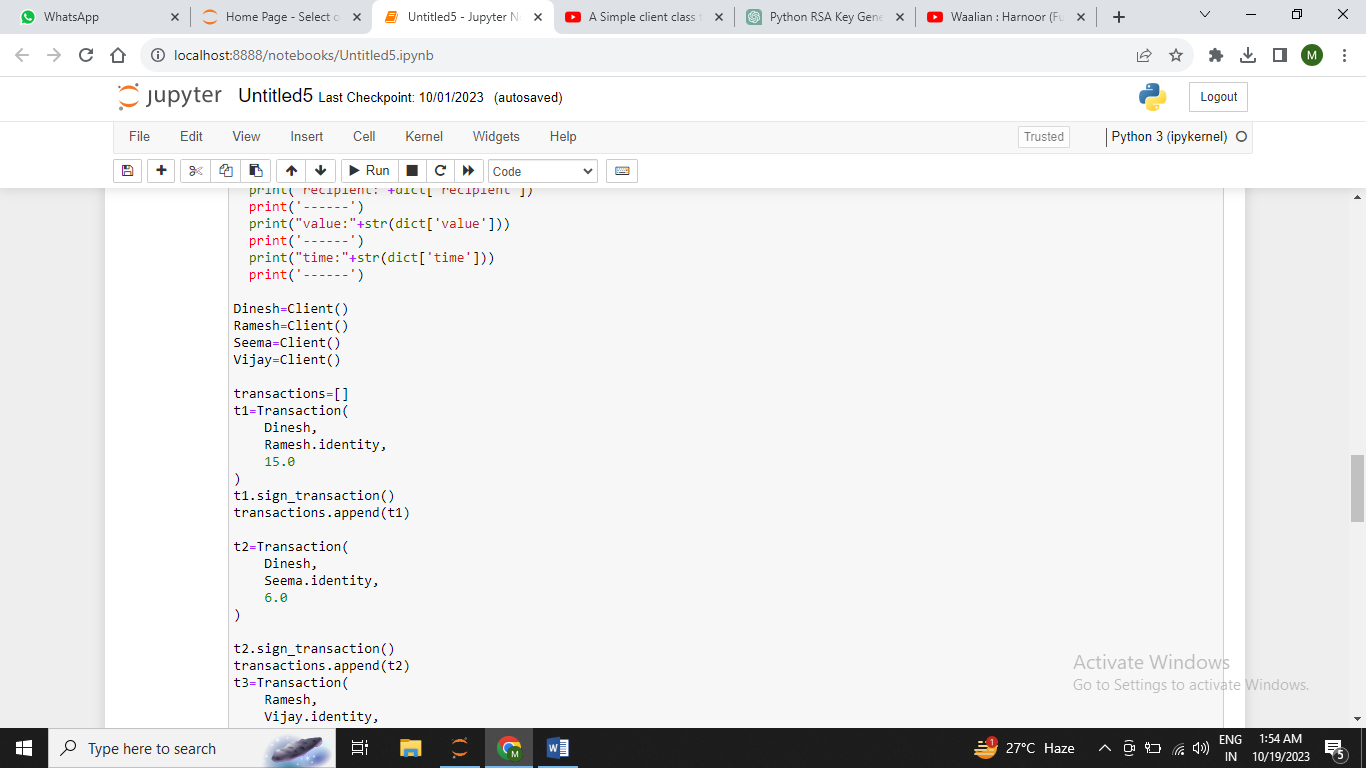
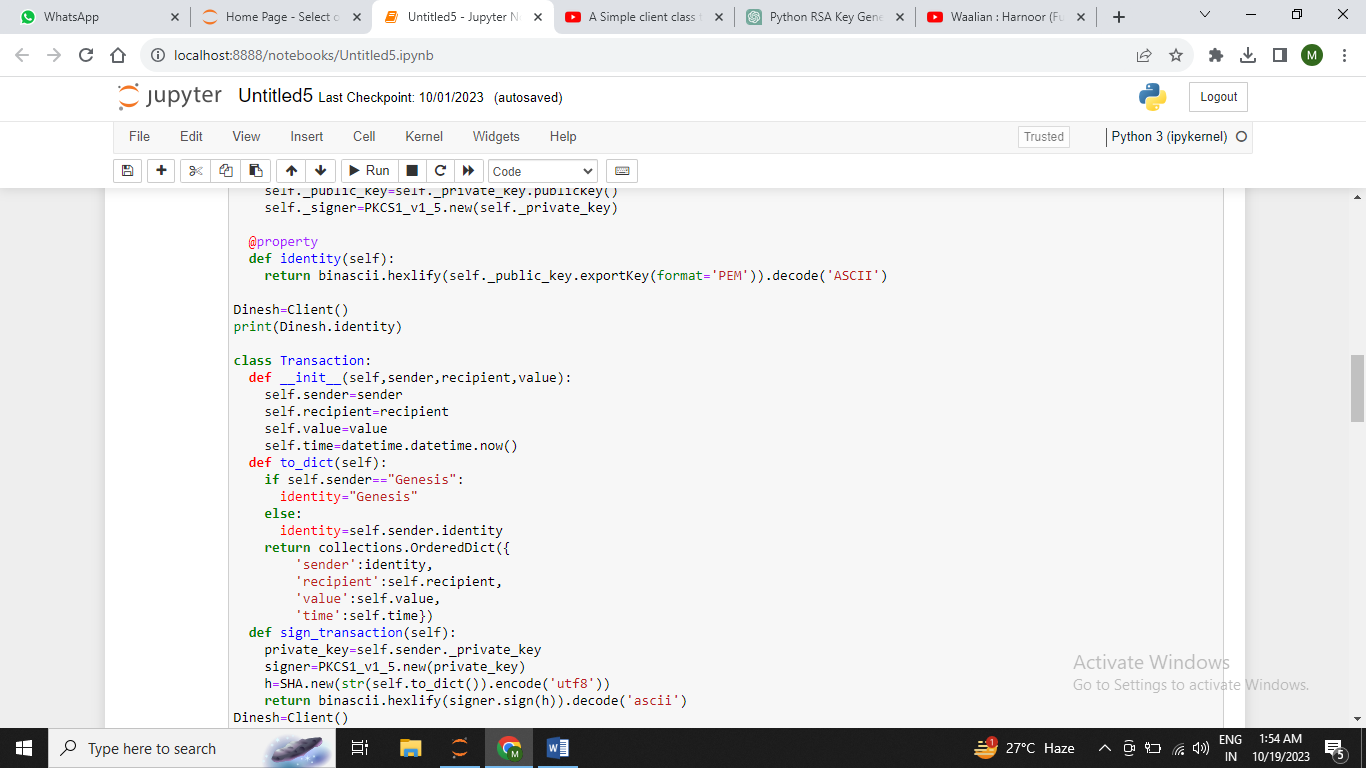
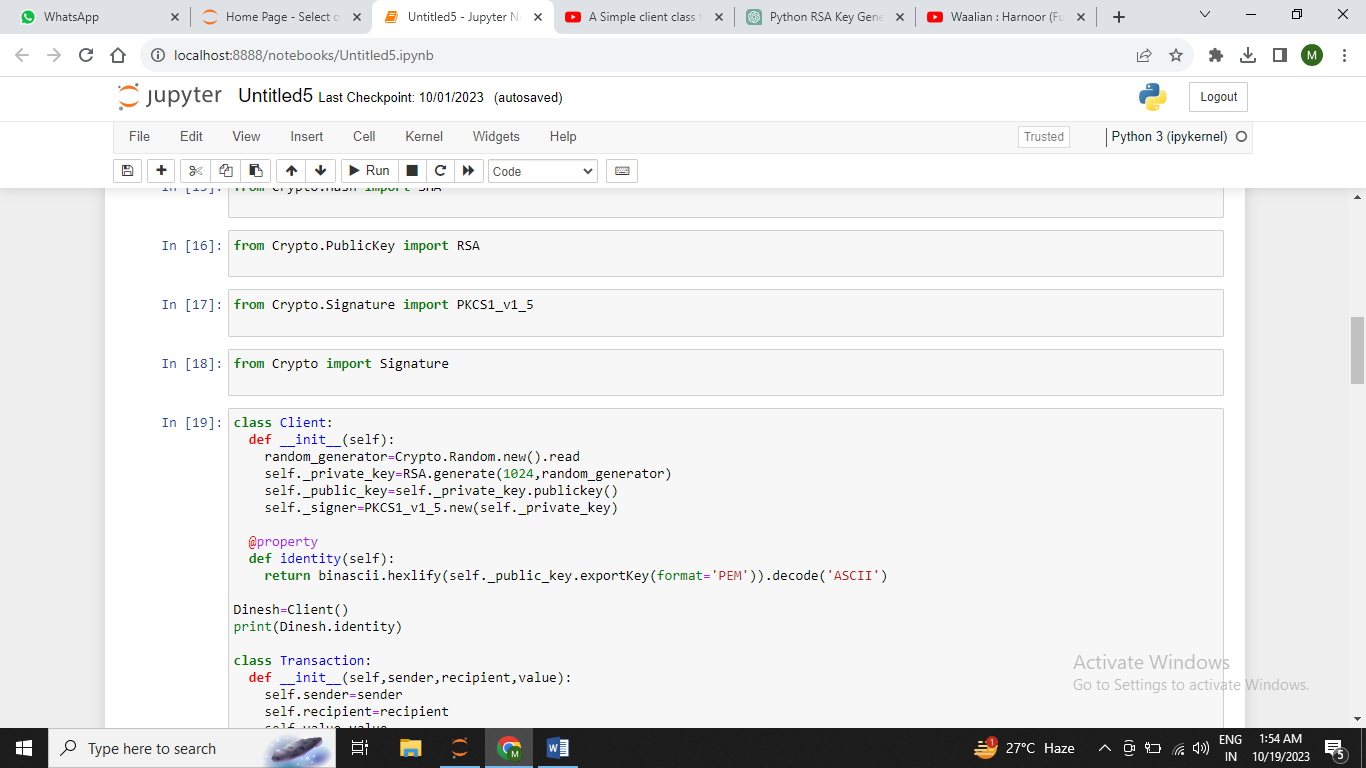
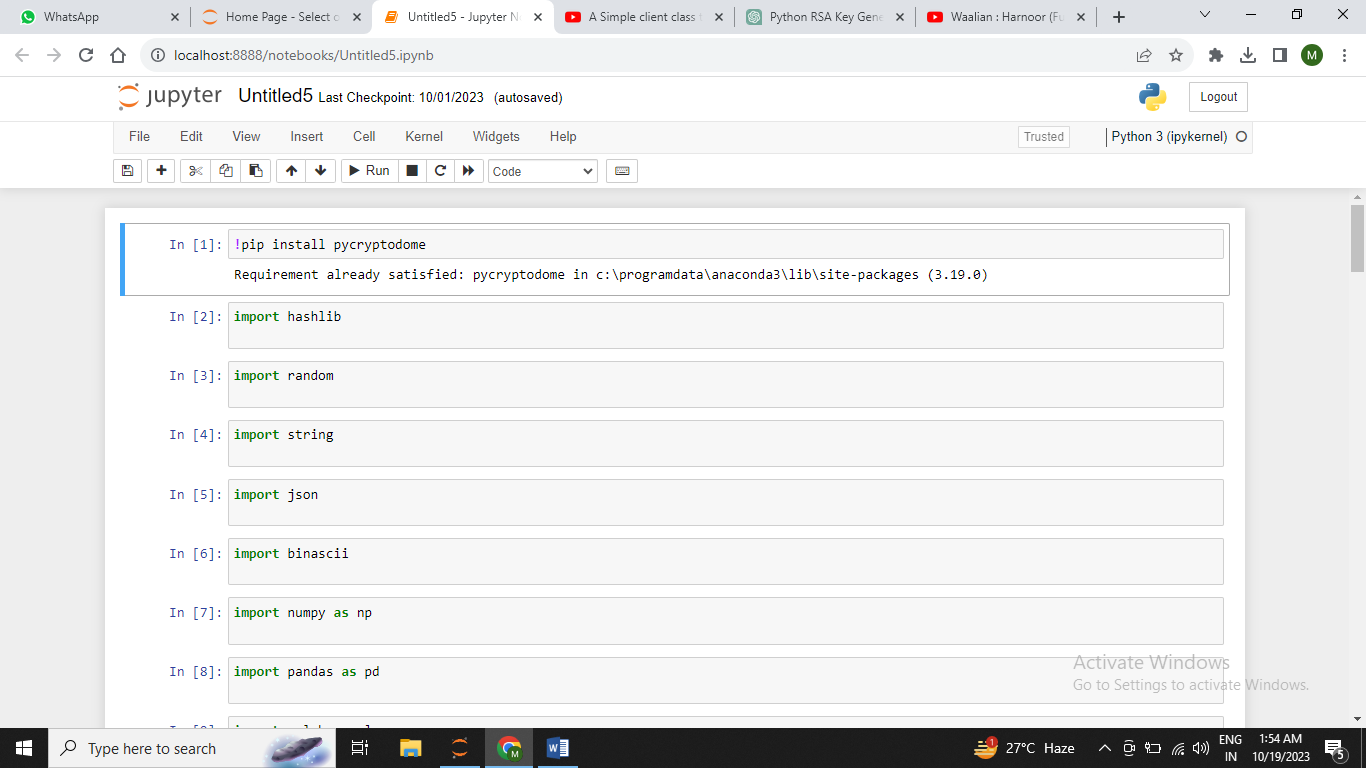
transactions.append(t10)

for transaction in transactions:

    display\_transaction(transaction)

    print('-------------')

Output:



**Practical 4**

Aim:Create a block chain genesis block and execute it

Codes:

!pip install pycryptodome

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

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto import Signature

class Client:

  def \_\_init\_\_(self):

    random\_generator=Crypto.Random.new().read

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

    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='PEM')).decode('ASCII')

Dinesh=Client()

print(Dinesh.identity)

class Transaction:

  def \_\_init\_\_(self,sender,recipient,value):

    self.sender=sender

    self.recipient=recipient

    self.value=value

    self.time=datetime.datetime.now()

  def to\_dict(self):

    if self.sender=="Genesis":

      identity="Genesis"

    else:

      identity=self.sender.identity

    return collections.OrderedDict({

        'sender':identity,

        'recipient':self.recipient,

        'value':self.value,

        'time':self.time})

  def sign\_transaction(self):

    private\_key=self.sender.\_private\_key

    signer=PKCS1\_v1\_5.new(private\_key)

    h=SHA.new(str(self.to\_dict()).encode('utf8'))

    return binascii.hexlify(signer.sign(h)).decode('ascii')

Dinesh=Client()

Ramesh=Client()

t=Transaction(

    Dinesh,

    Ramesh.identity,

    5.0

)

signature=t.sign\_transaction()

print(Signature)

print("=============Multiple Transaction=============")

def display\_transaction(transaction):

  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('------')

class Block:

  def \_\_init\_\_(self):

    self.verified\_transactions=[]

    self.previous\_block\_hash=""

    self.Nonce=""

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("==========")

Dinesh=Client()

t0=Transaction(

    "Genesis",

    Dinesh.identity,

    500.0

)

block0=Block()

Nonce=None

block0.verified\_transactions.append(t0)

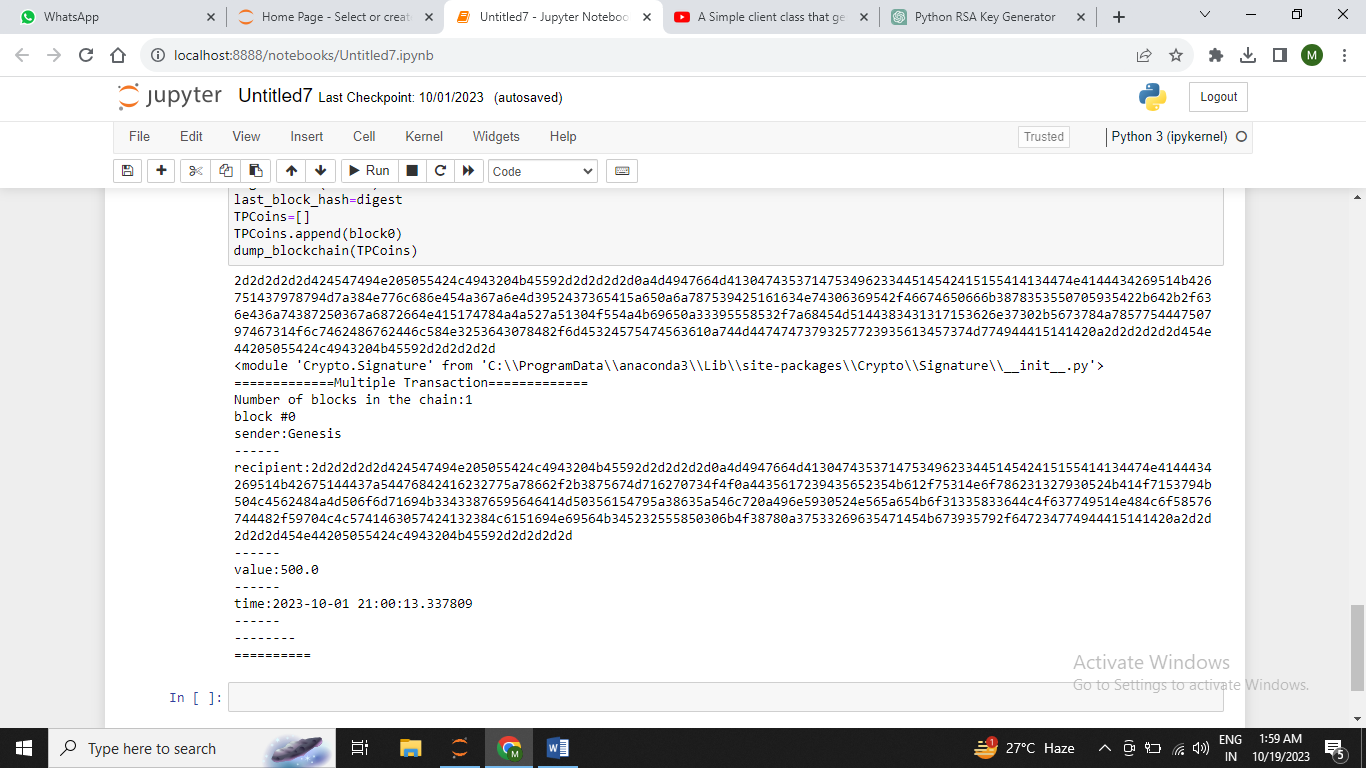
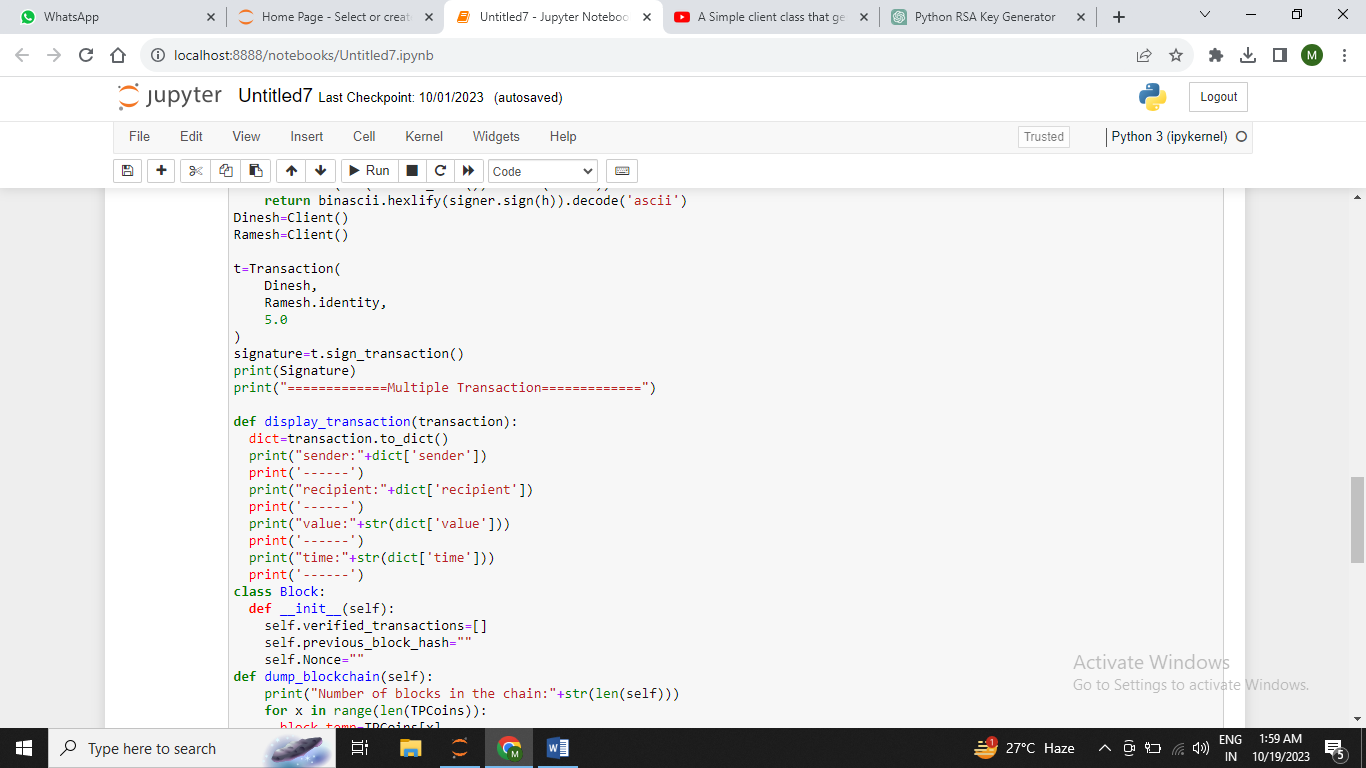
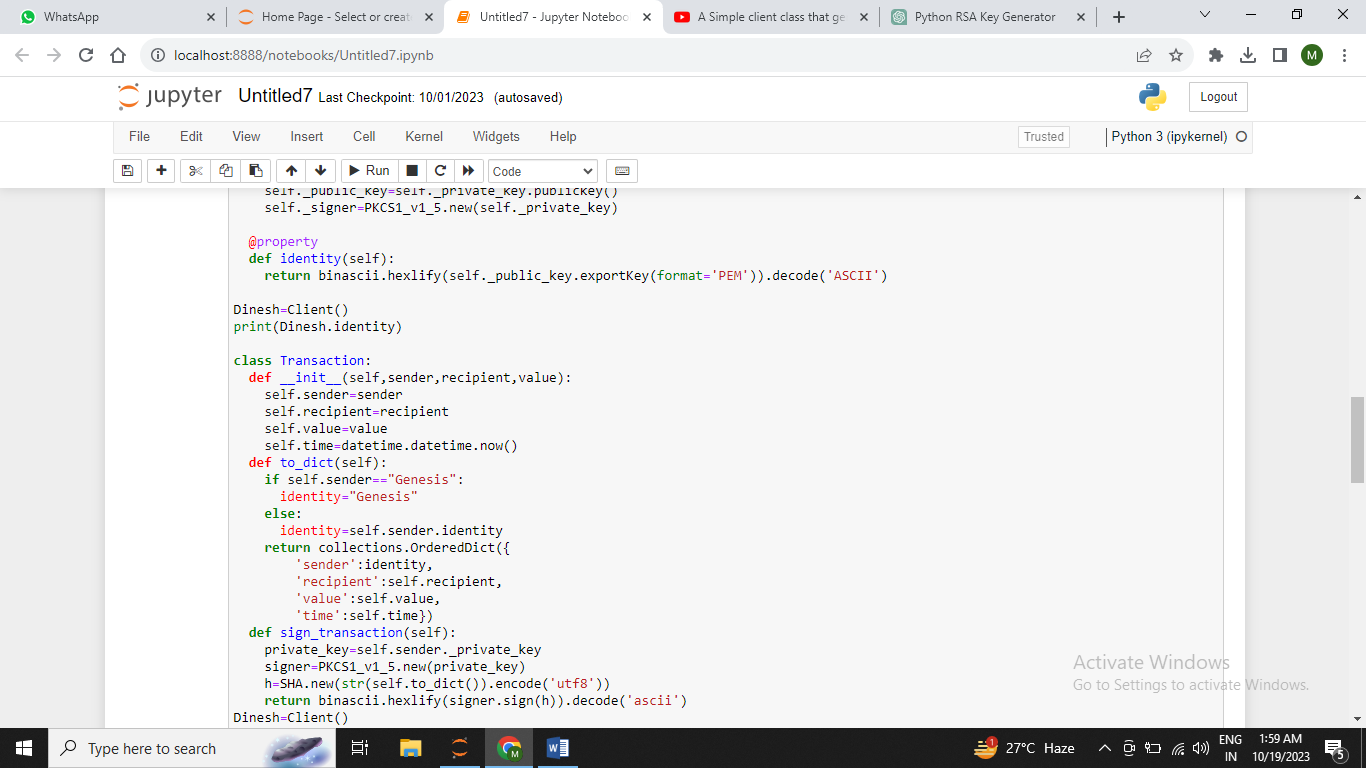
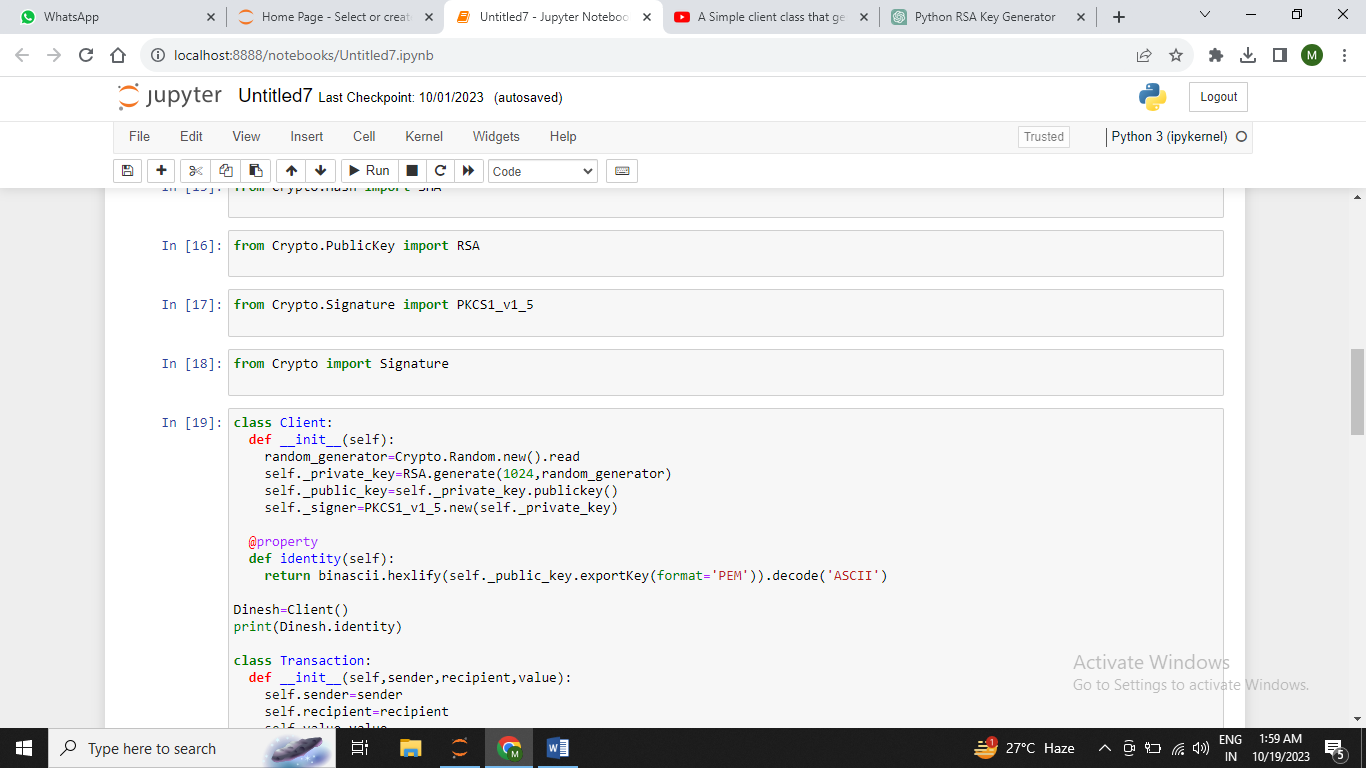
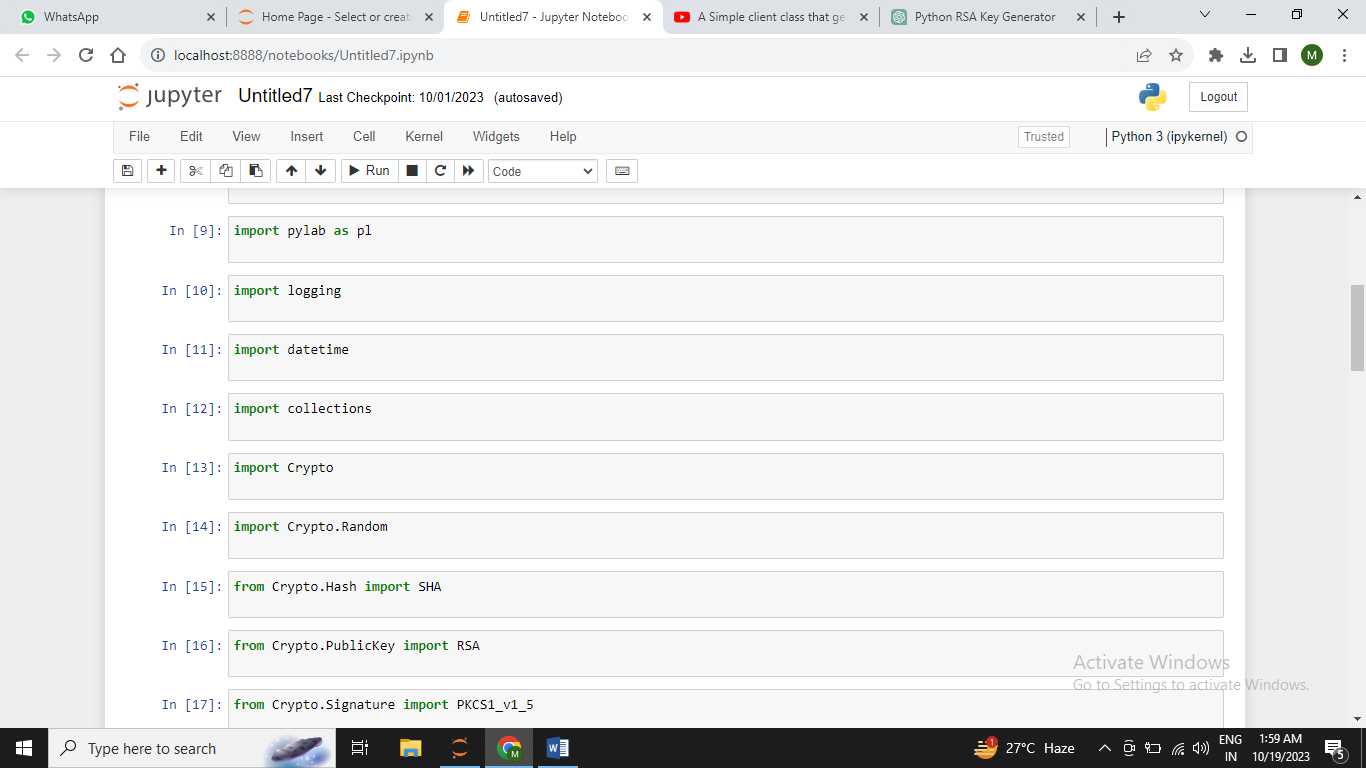
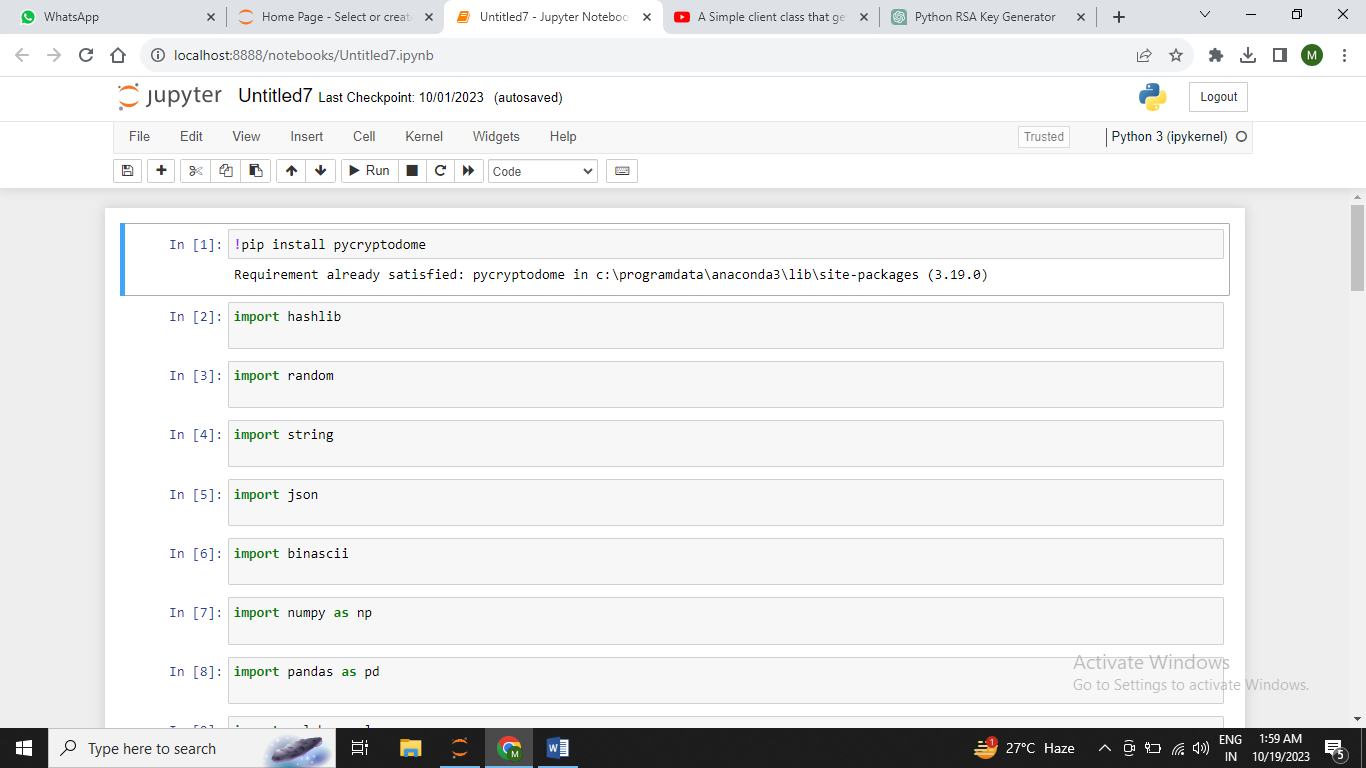
digest=hash(block0)

last\_block\_hash=digest

TPCoins=[]

TPCoins.append(block0)

dump\_blockchain(TPCoins)

Output:****

**Practical 5**

Aim: Creating a mining function and testing it

Codes:

!pip install pycryptodome

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

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

def sha256(message):

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

def mine(message,difficulty=1):

  assert difficulty>=1

  prefix='1' \* difficulty

  for i in range(1000):

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

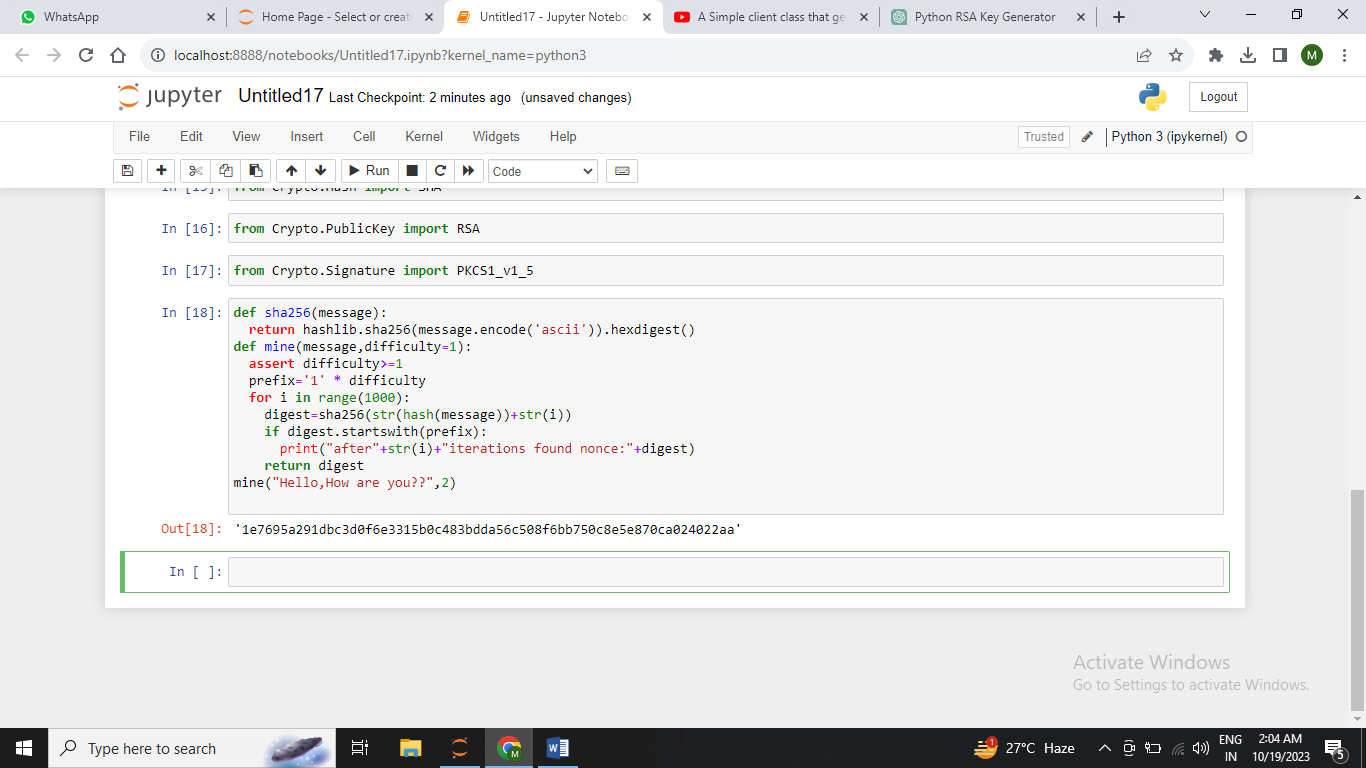
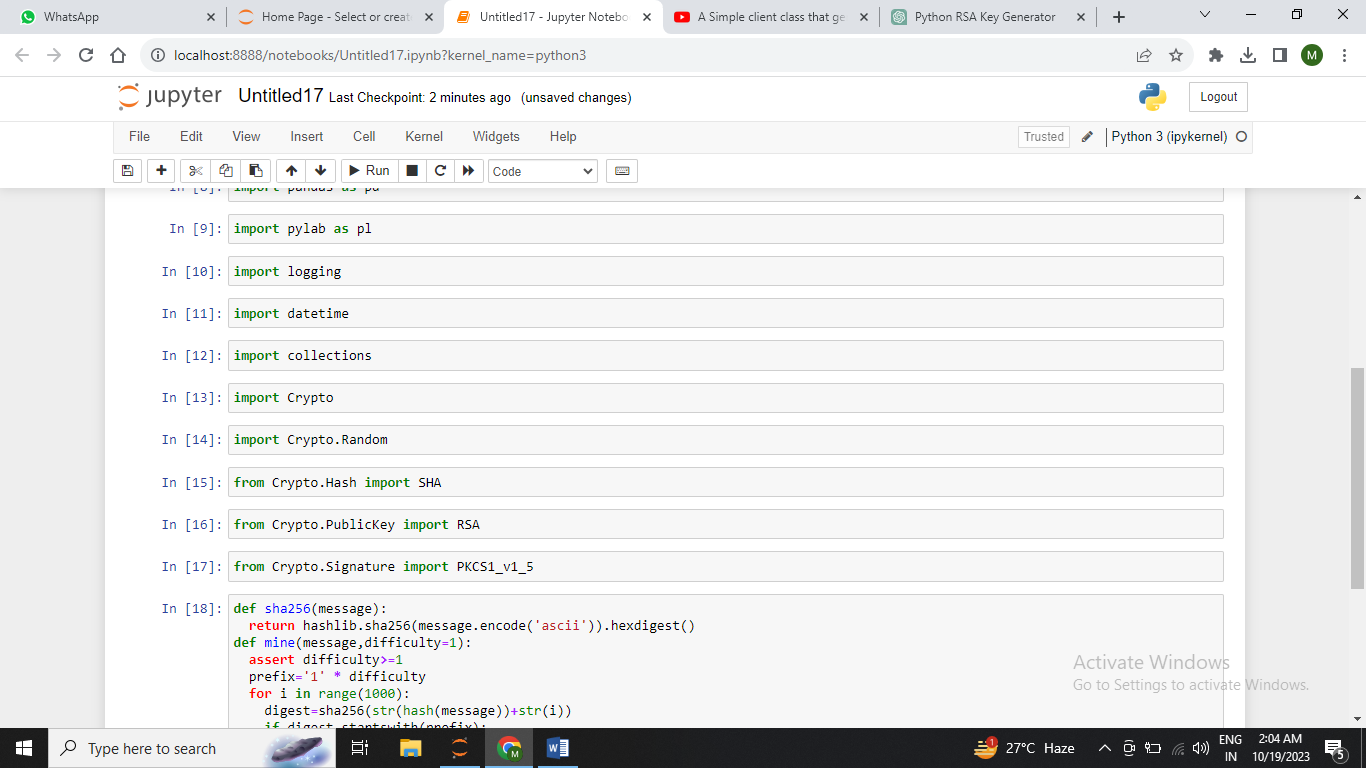
    if digest.startswith(prefix):

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

    return digest

mine("Hello,How are you??",2)

Output:



**Practical 6**

**Aim : Creating a smart contract on the Ethereum blockchain using Remix**

**Code :** // SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SimpleStorage {

uint256 public storedData;

constructor() {

storedData = 0;

}

function set(uint256 newValue) public {

storedData = newValue;

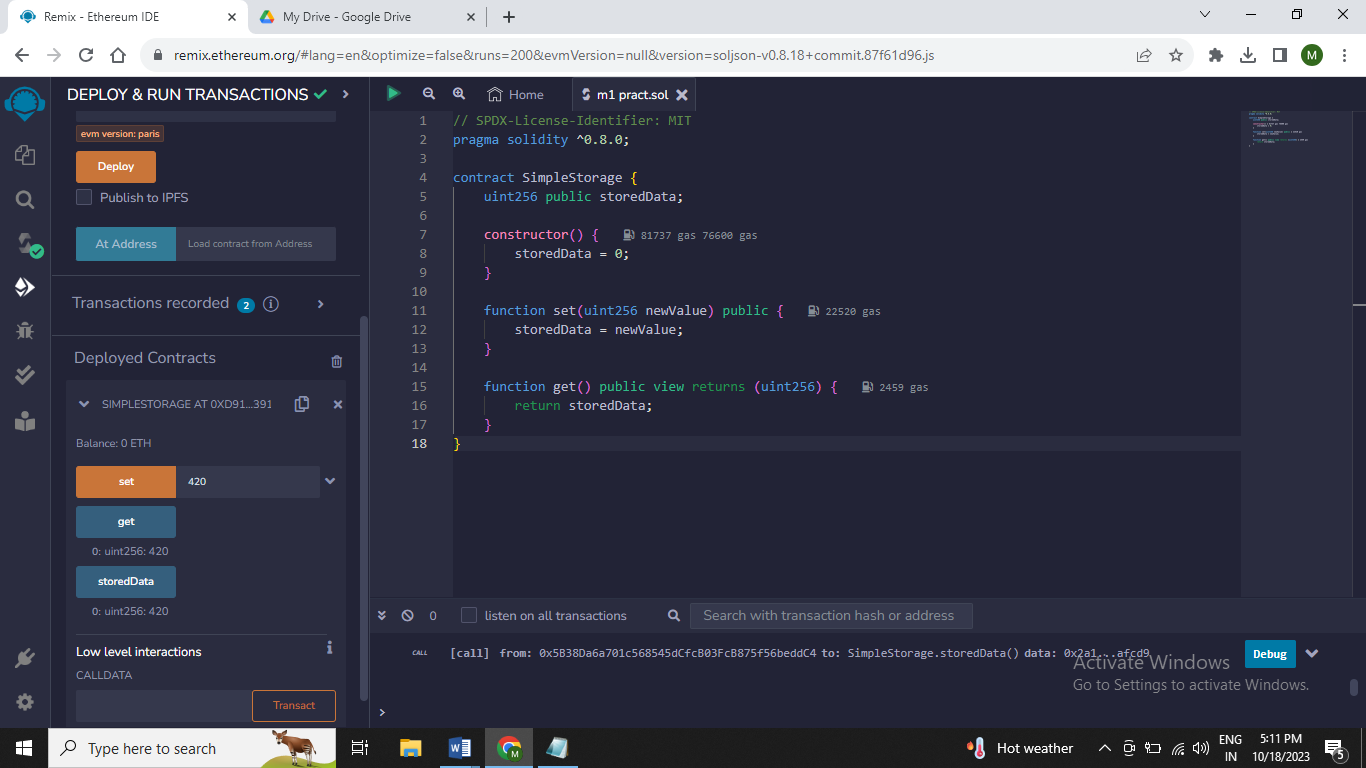
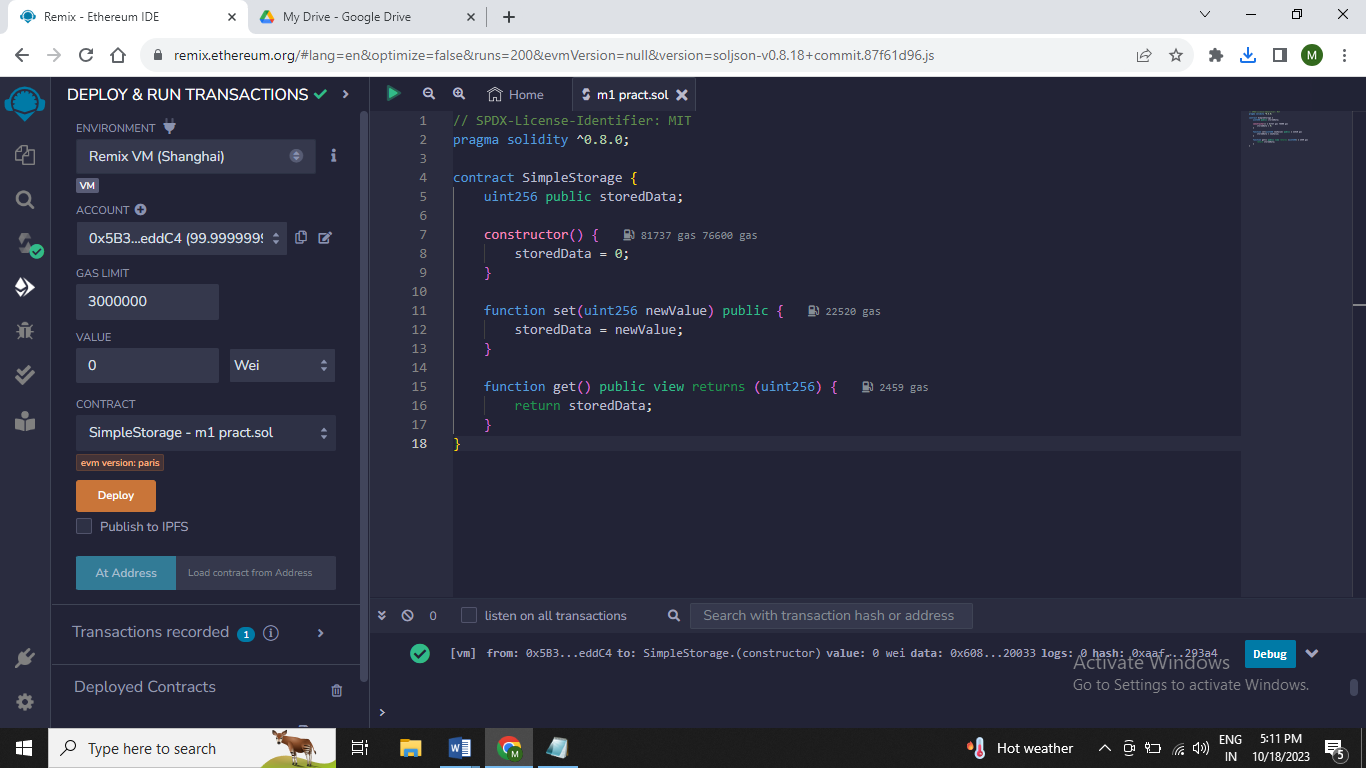
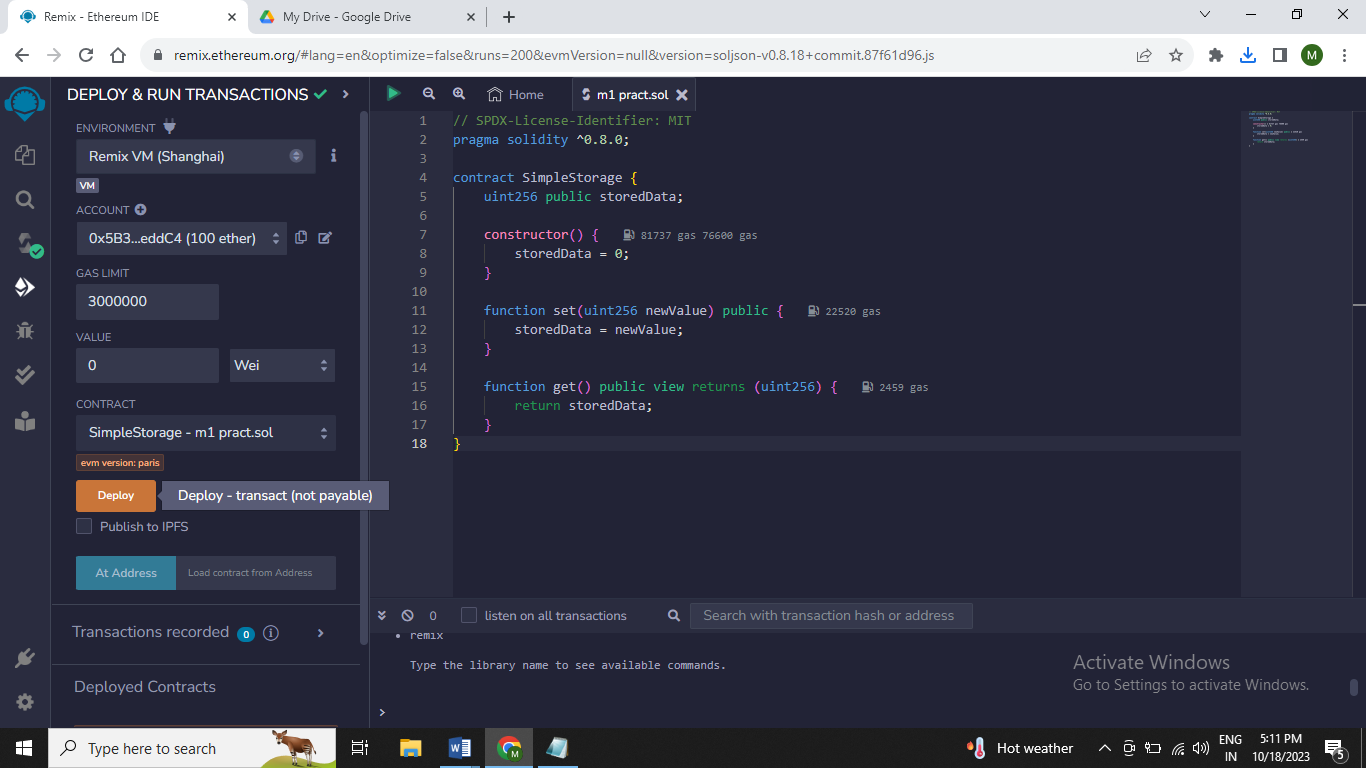
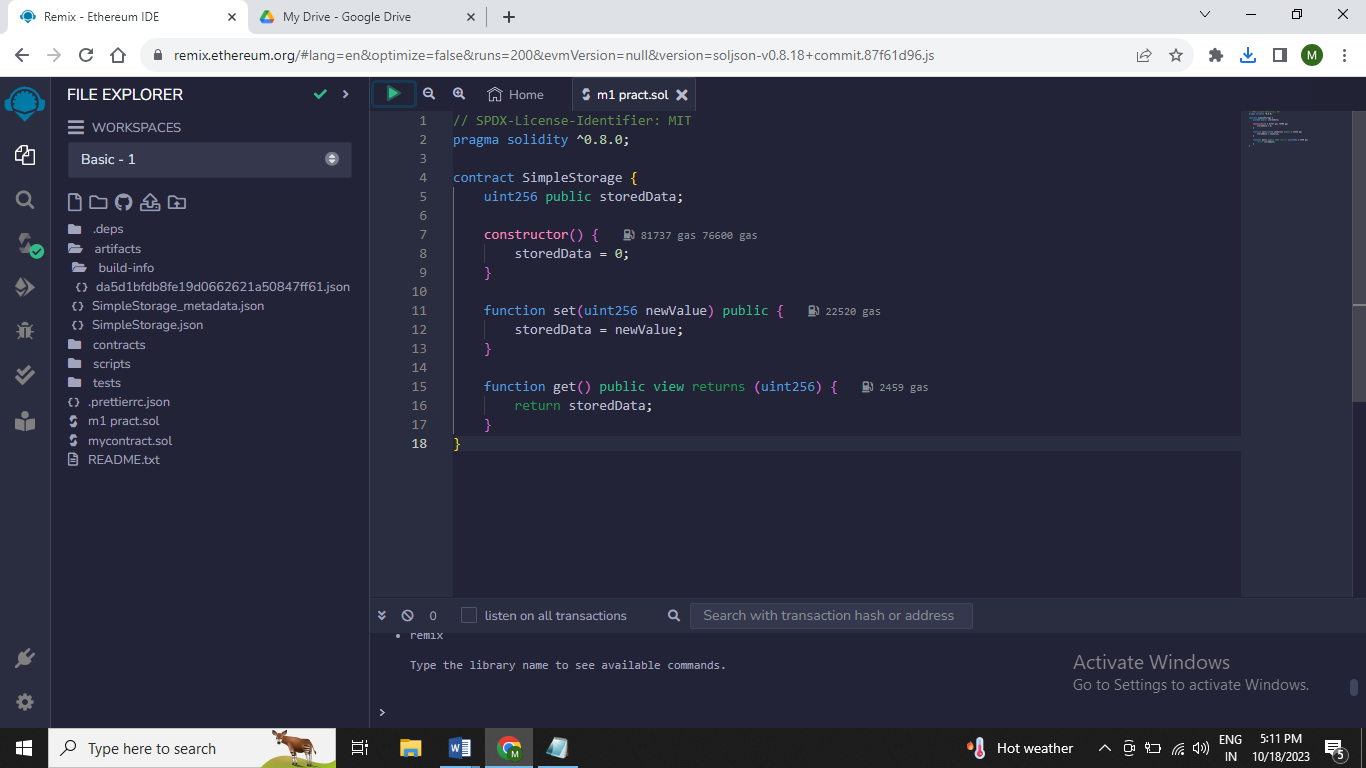
}

function get() public view returns (uint256) {

return storedData;

}

}

Output : ****

**Practical 7**

**Aim : Creating a variable on the blockchain using Solidity and deploying it as contract on the Ethereum blockchain using Remix**

**Code :**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SimpleStorage {

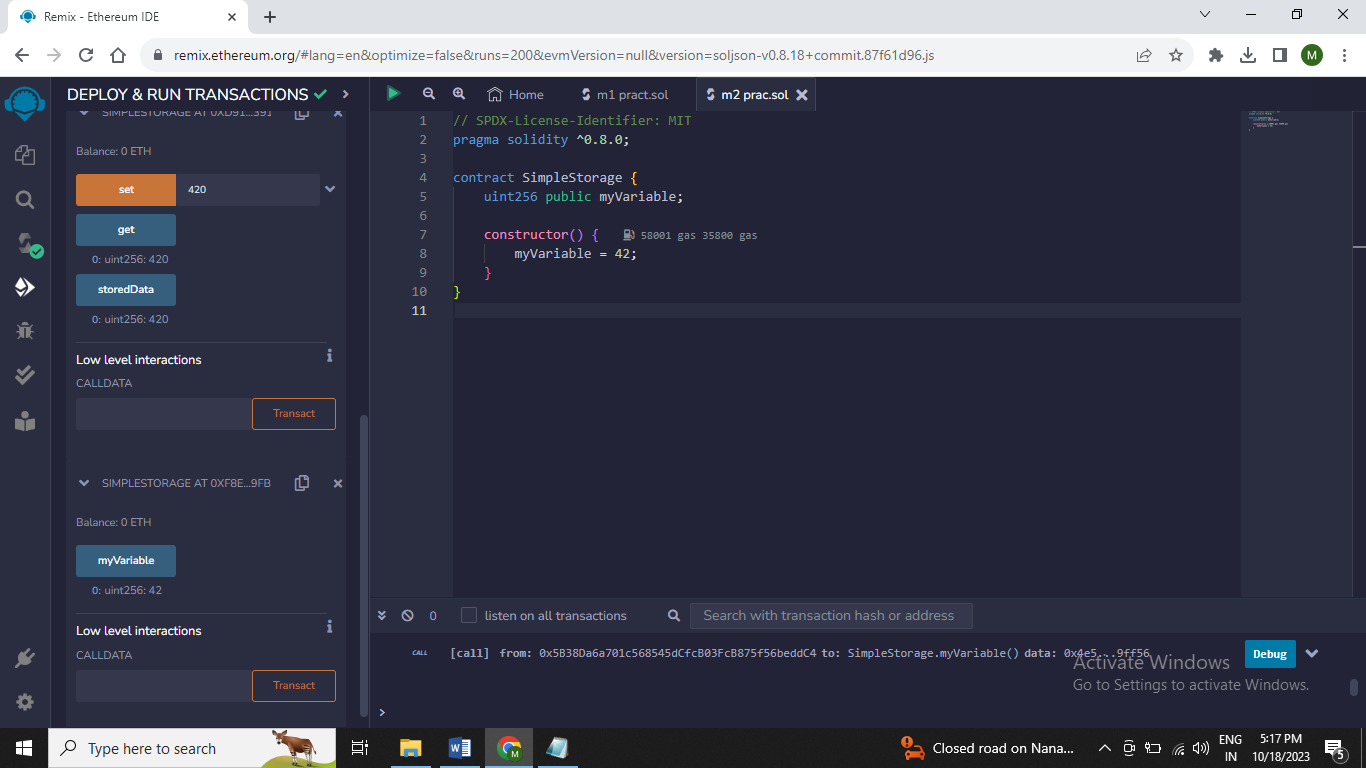
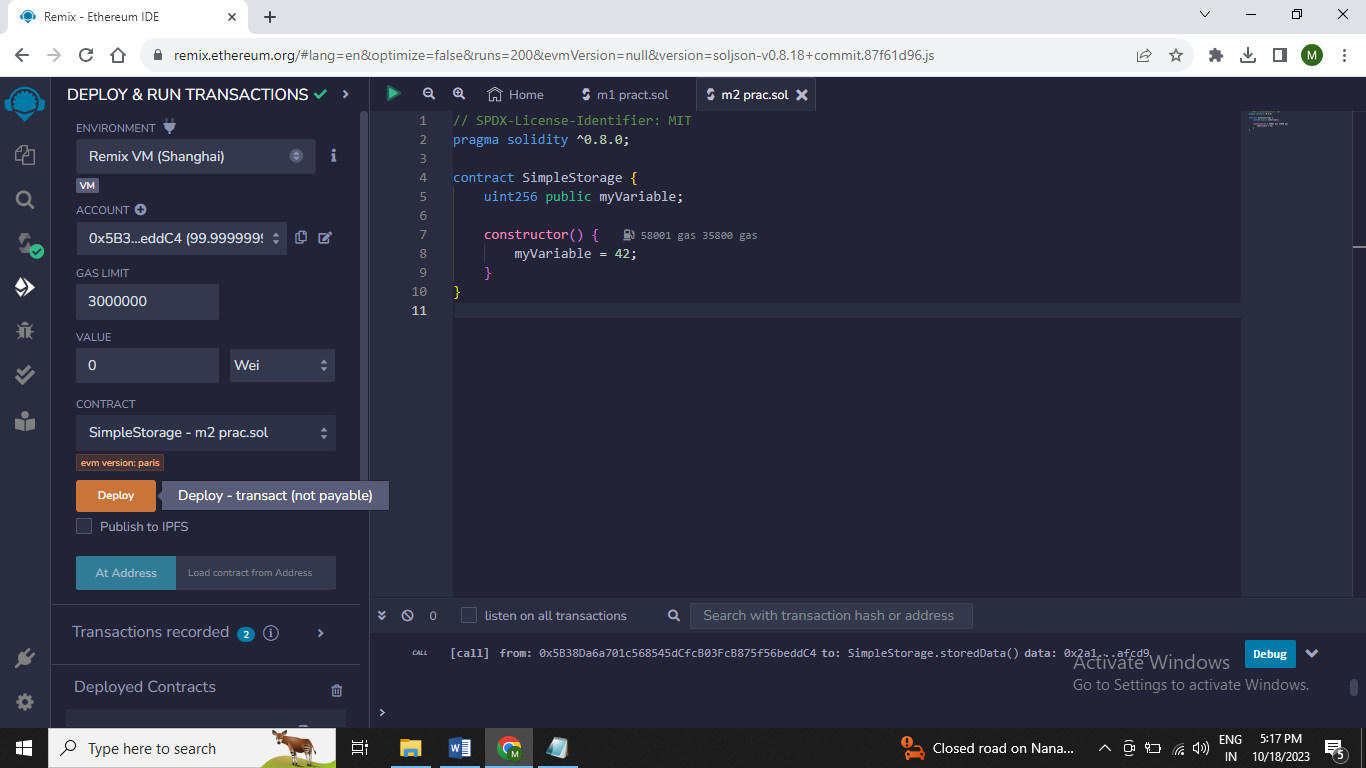
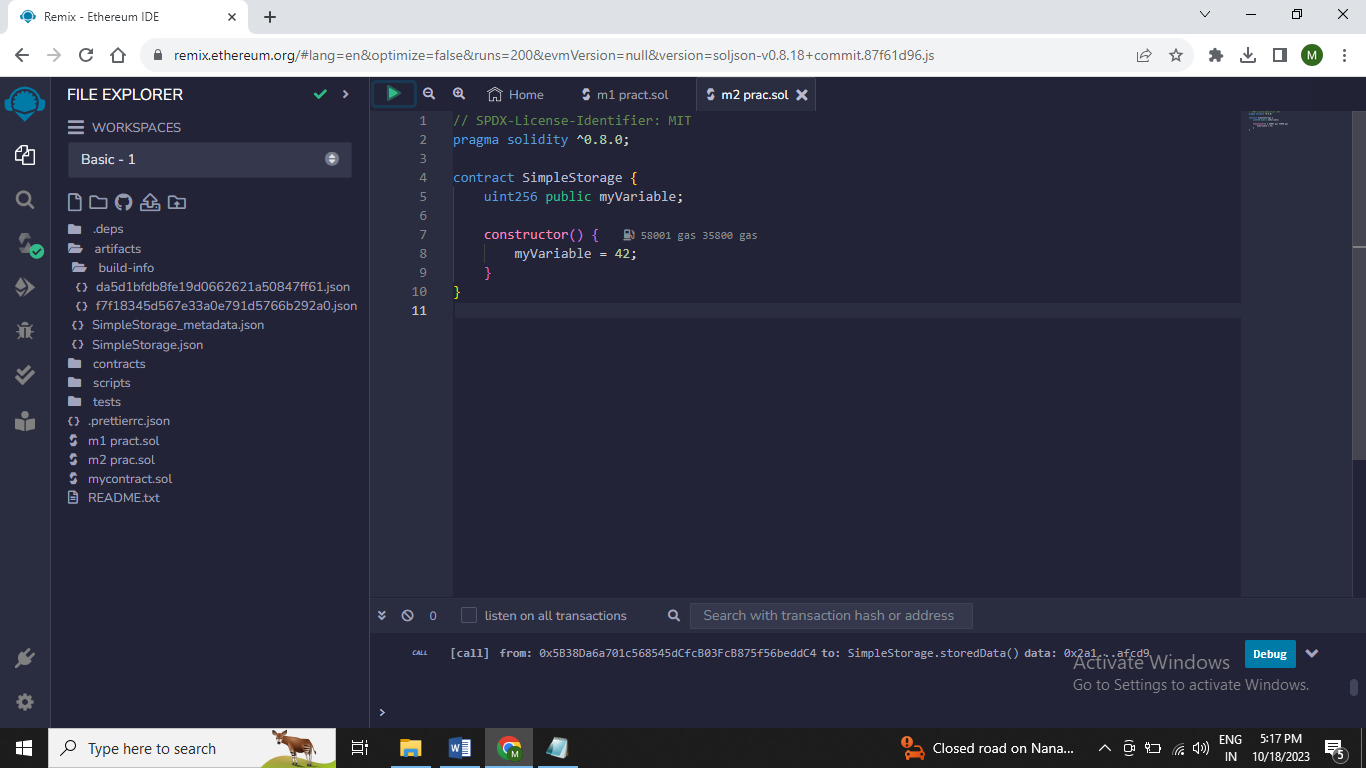
uint256 public myVariable;

constructor() {

myVariable = 42;

}

}

Output: ****

**Practical 8**

**Aim: Creating an array on the blockchain using Solidity and deploying it as contract on the Ethereum blockchain using Remix**

**Code :**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract ArrayExample {

uint256[] public myArray;

constructor() {

myArray.push(42);

}

function addValue(uint256 \_value) public {

myArray.push(\_value);

}

function getArrayLength() public view returns (uint256) {

return myArray.length;

}

function getValueAtIndex(uint256 index) public view returns (uint256) {

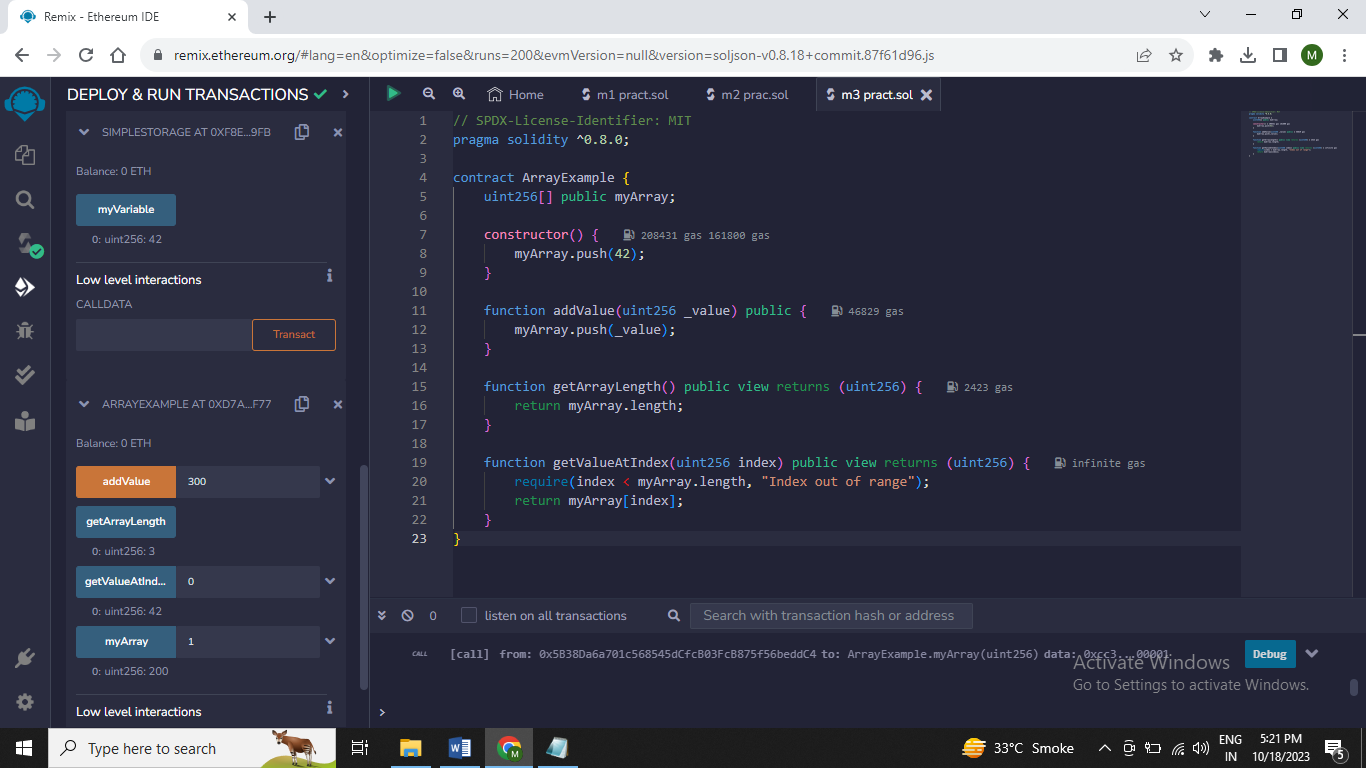
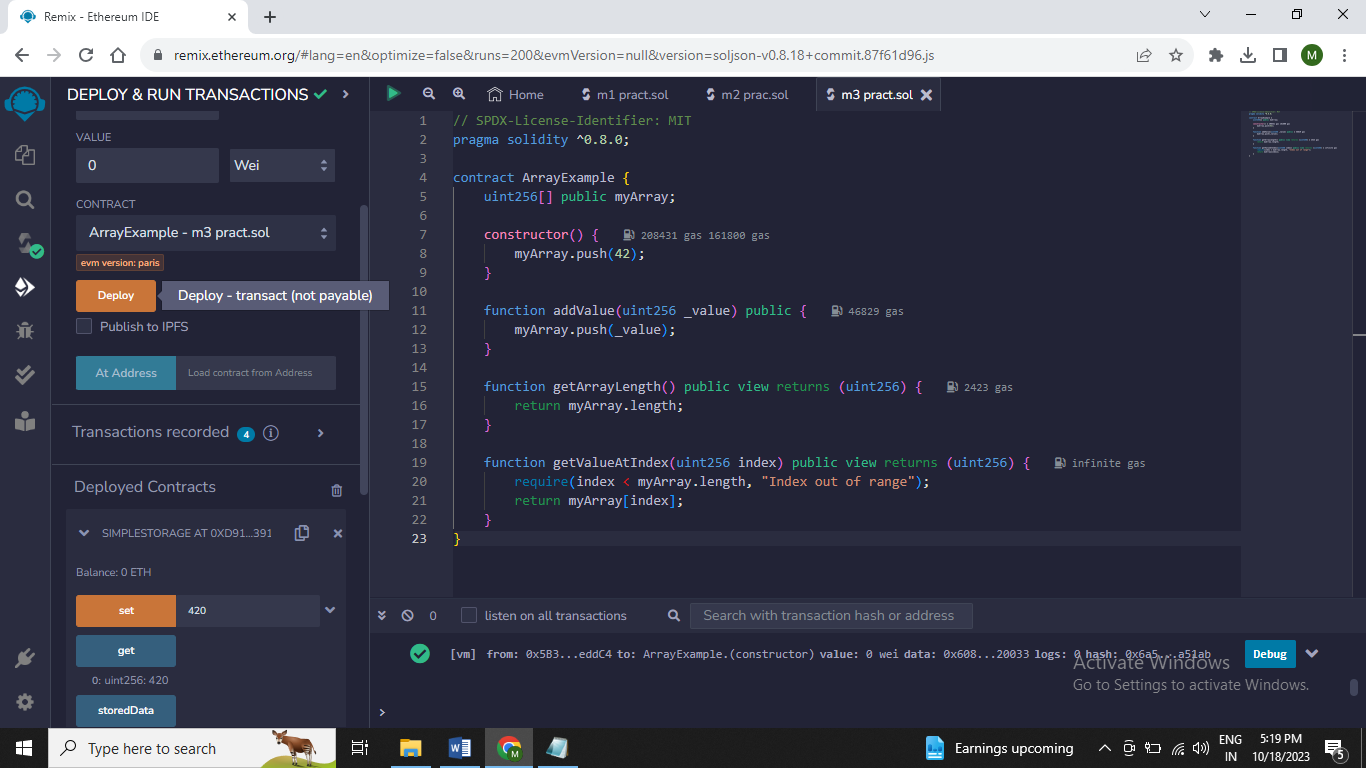
require(index < myArray.length, "Index out of range");

return myArray[index];

}

}

Output:

****

**Practical 9**

**Aim: Creating a for loop on the blockchain using Solidity and deploying it as contract on the Ethereum blockchain using Remix**

**Code :** // SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract LoopExamplevalues {

uint256[] public values;

constructor() {

values = new uint256[](0);

}

function initializeArray(uint256[] memory initialData) public {

values = initialData;

}

function incrementArray() public {

for (uint256 i = 0; i < values.length; i++) {

values[i] = values[i] + 1;

}

}

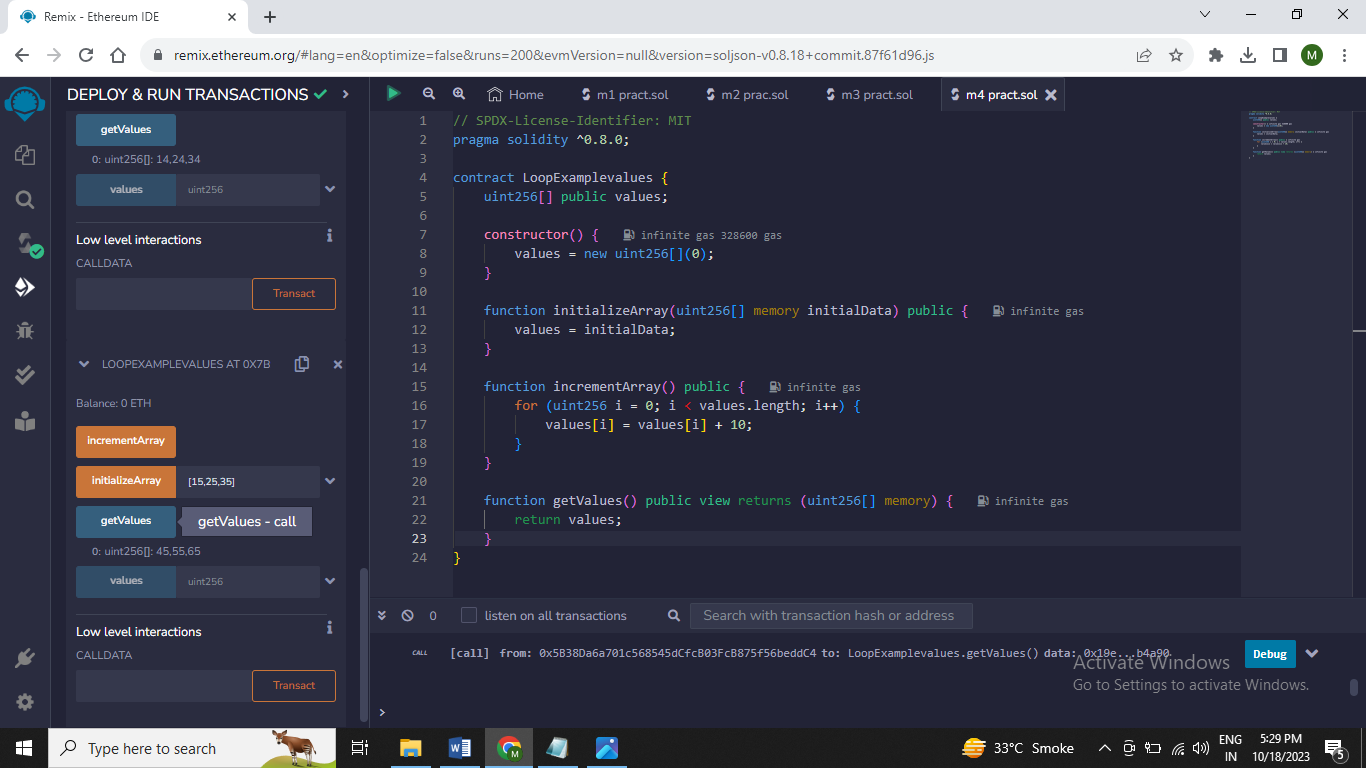
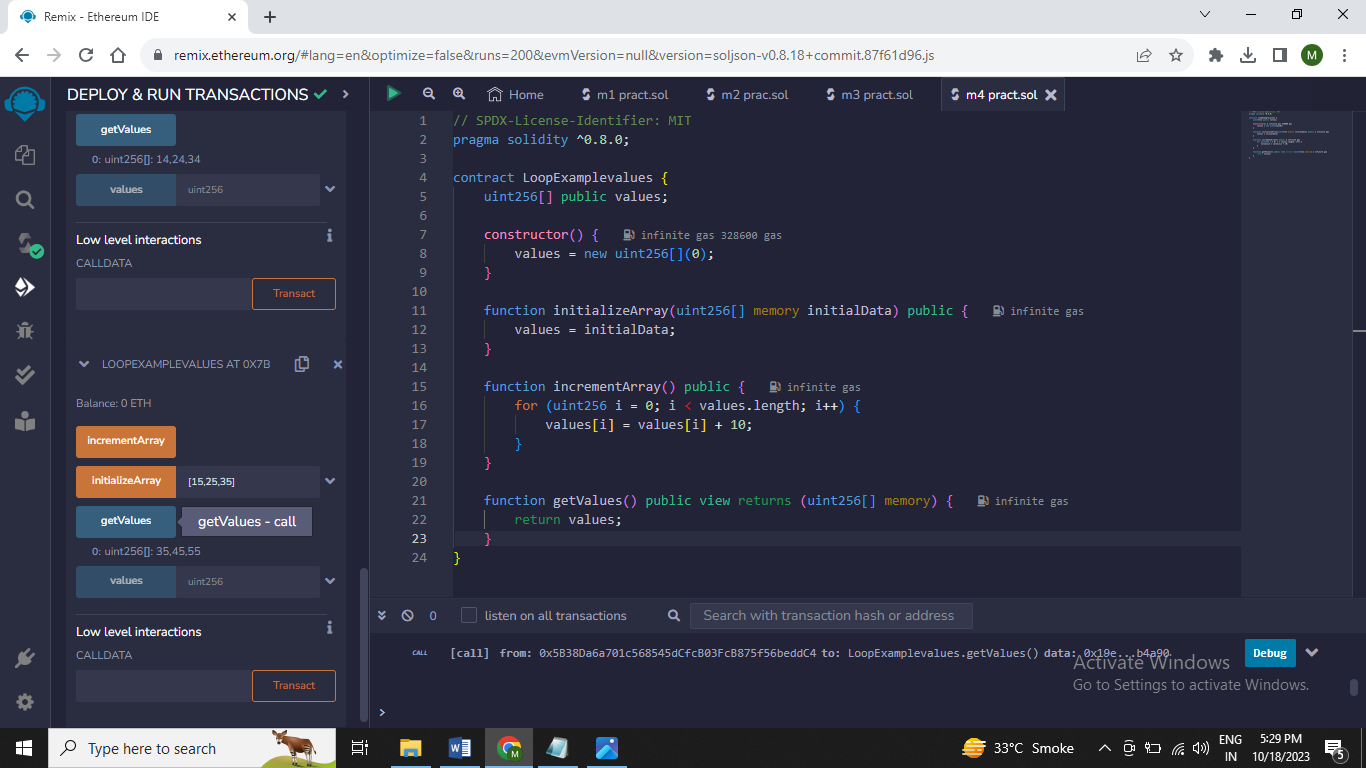
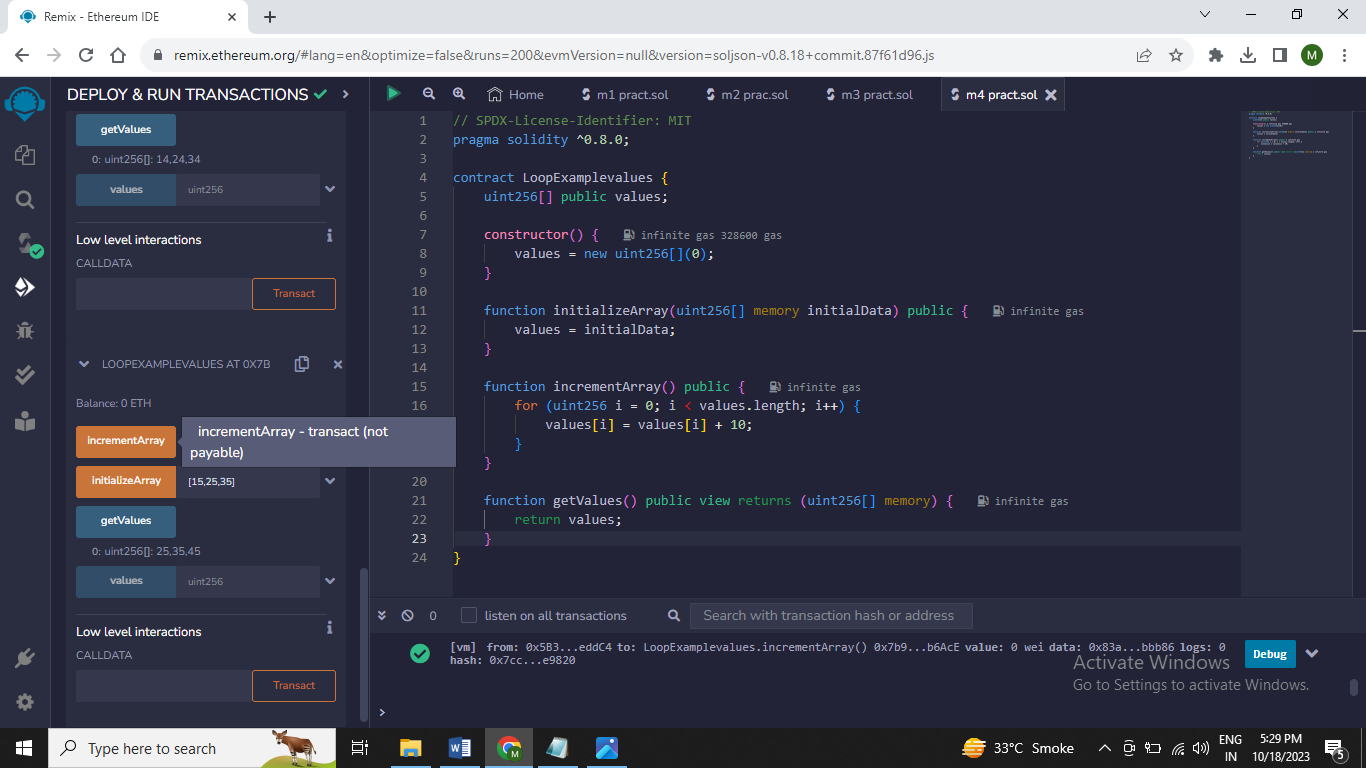
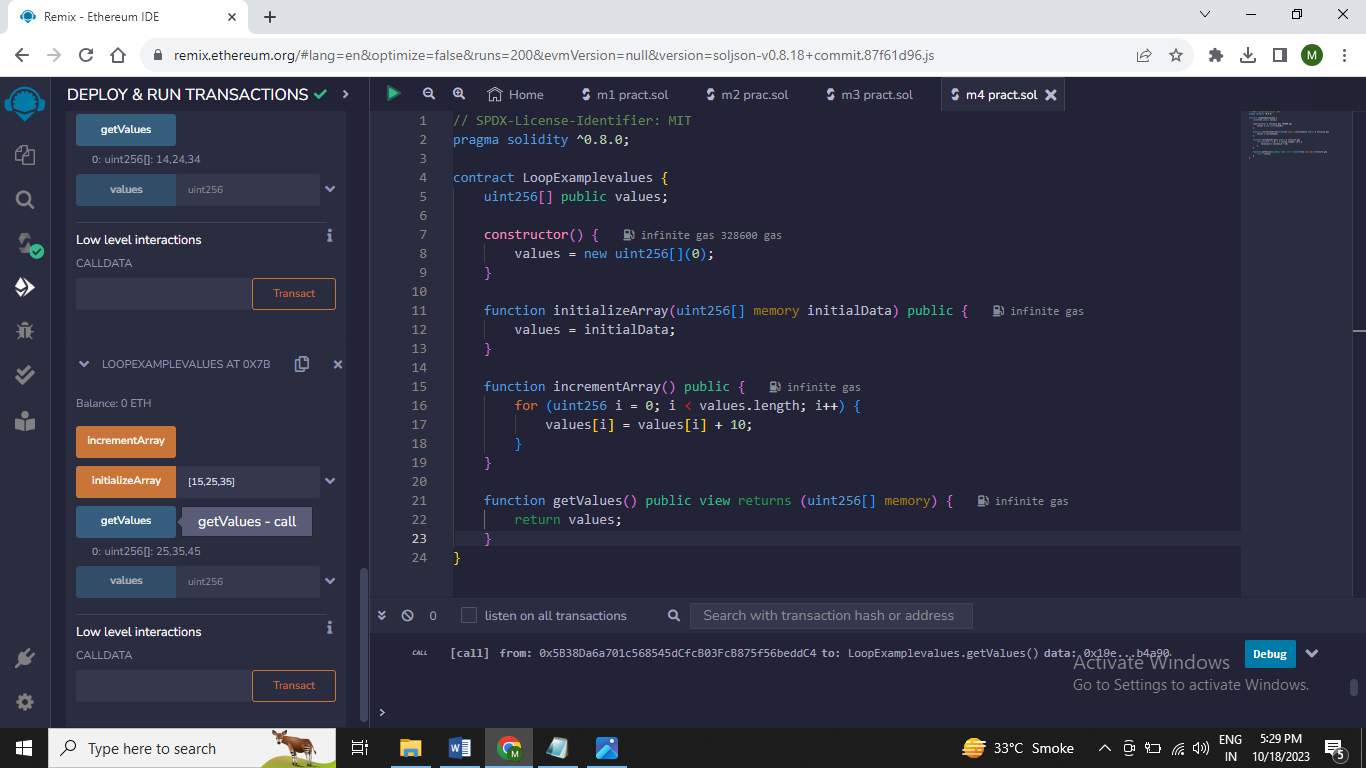
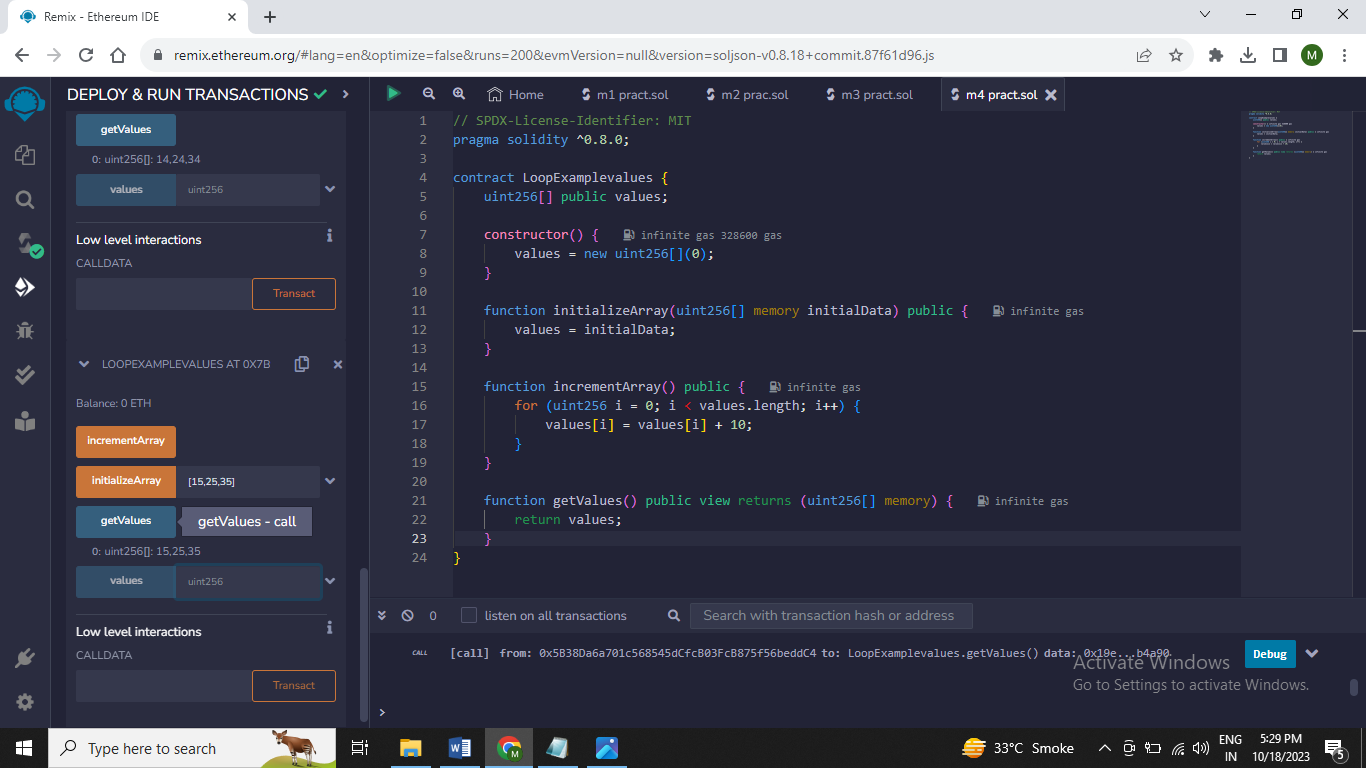
function getValues() public view returns (uint256[] memory) {

return values;

}

}

Output:

****

**Practical 10**

**Aim: Creating Struct on the blockchain using Solidity and deploying it as contract on the Ethereum blockchain using Remix**

Code : // SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract EmployeeRegistrymul {

struct Employee {

string name;

uint256 salary;

}

mapping(address => Employee) public employees;

function addEmployee(address \_address, string memory \_name, uint256 \_salary) public {

require(\_address != address(0), "Invalid address");

employees[\_address] = Employee(\_name, \_salary);

}

function updateSalary(address \_address, uint256 \_newSalary) public {

require(\_address != address(0), "Invalid address");

Employee storage employee = employees[\_address];

employee.salary = \_newSalary;

}

function getEmployeeSalary(address \_address) public view returns (uint256) {

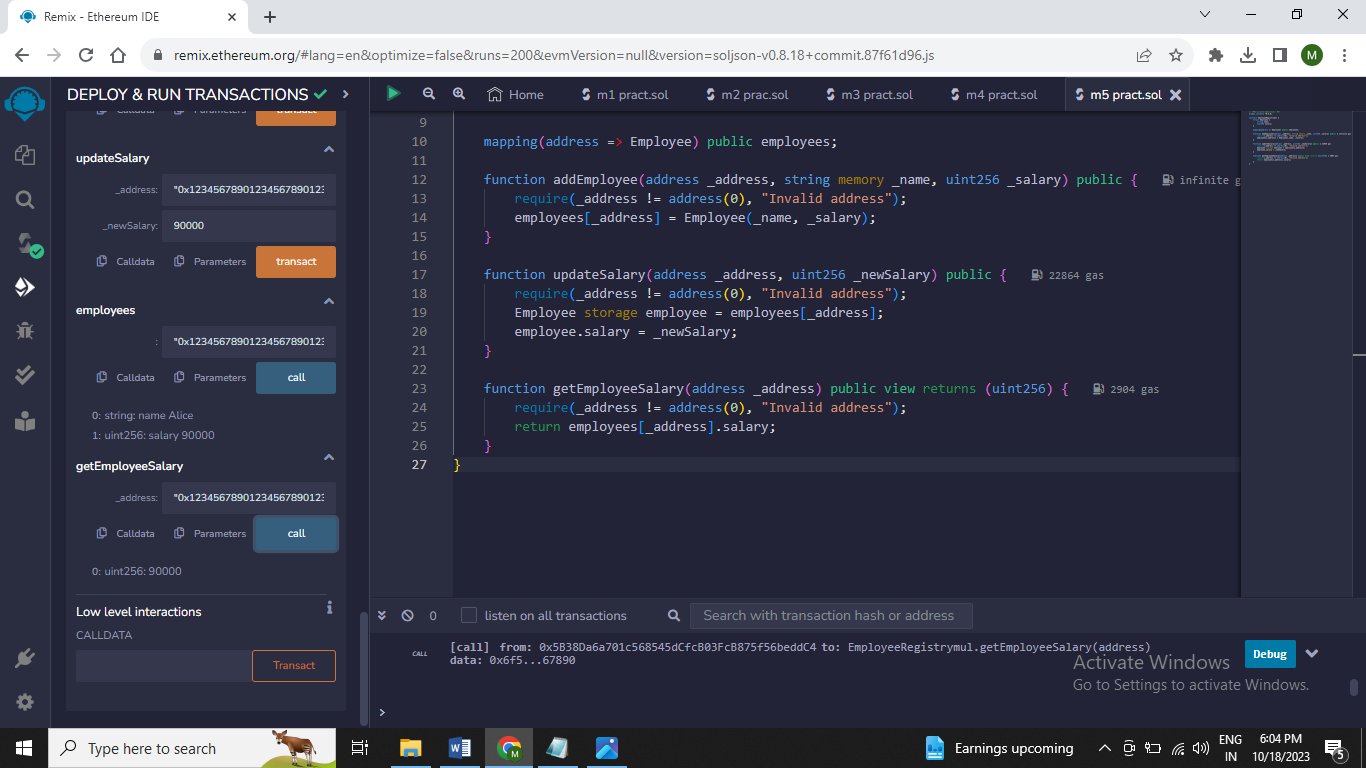
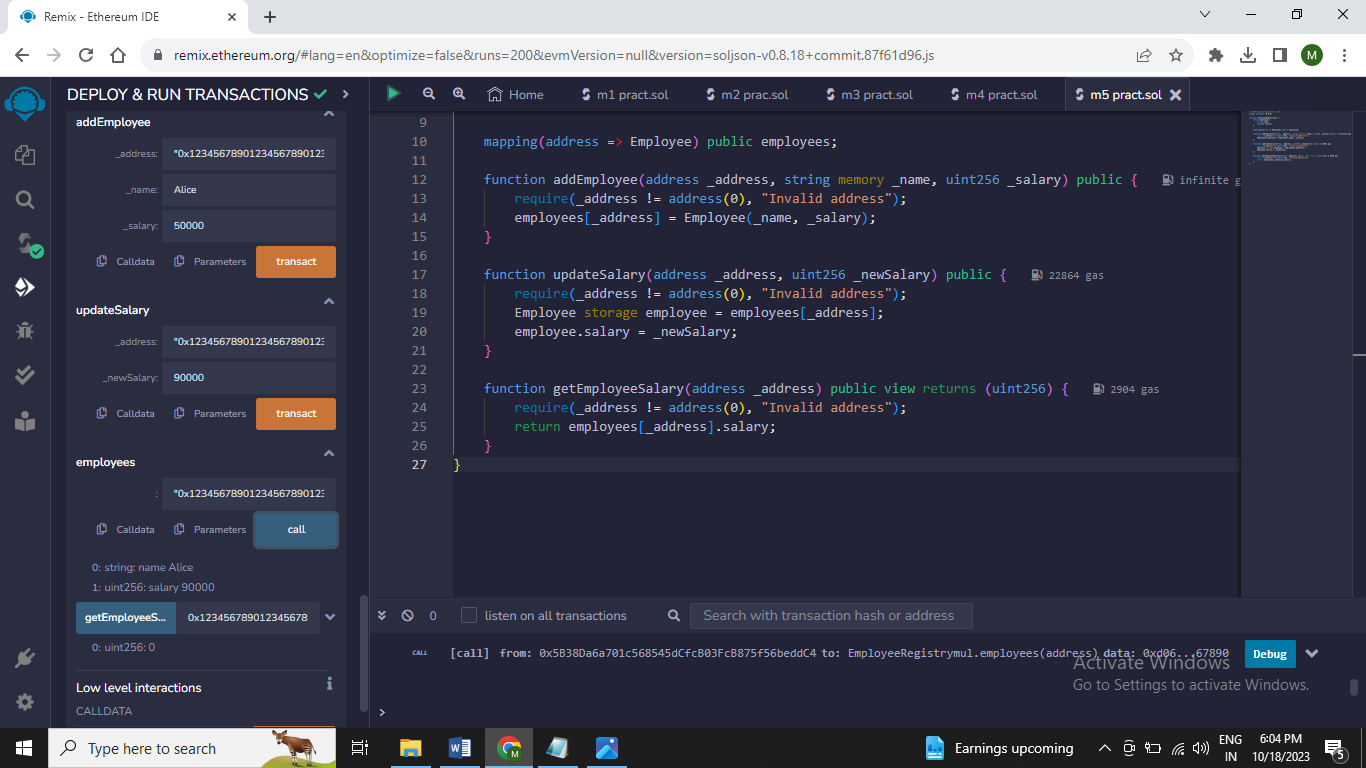
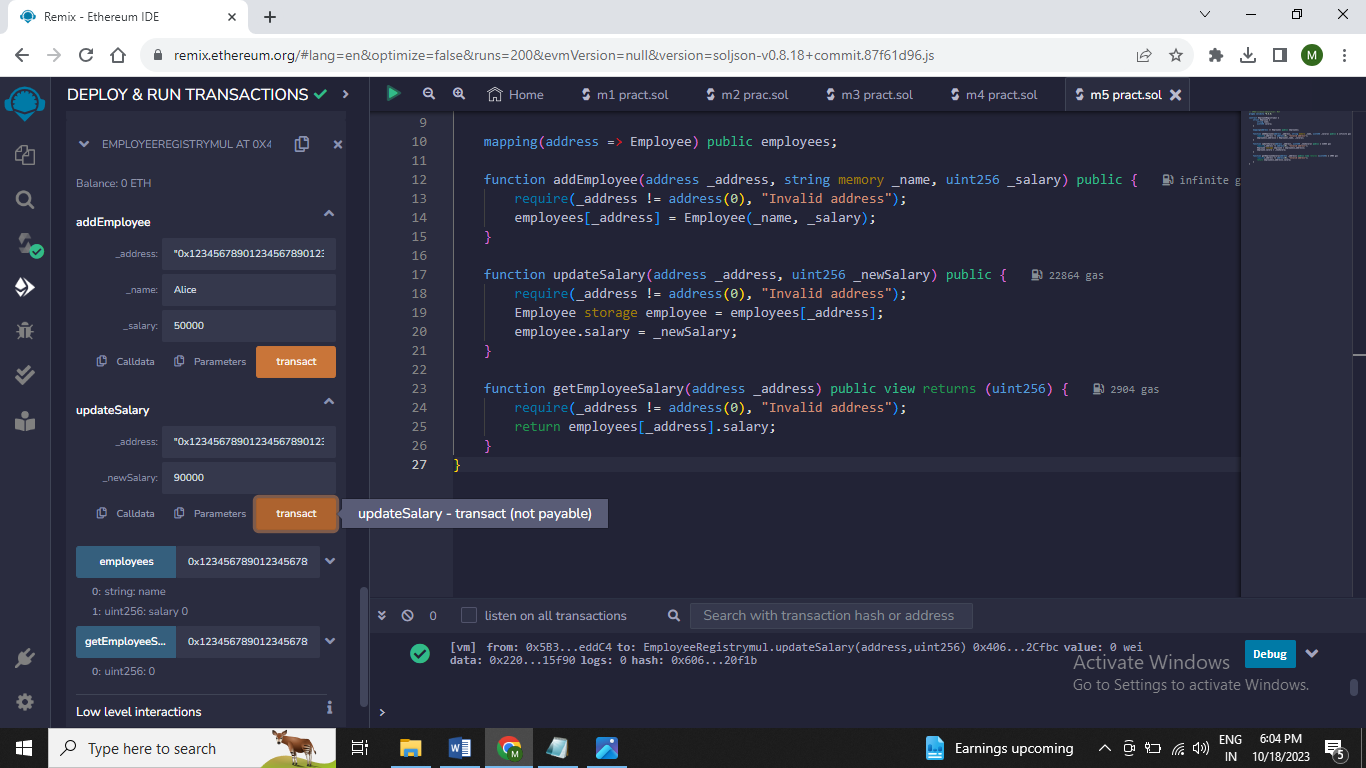
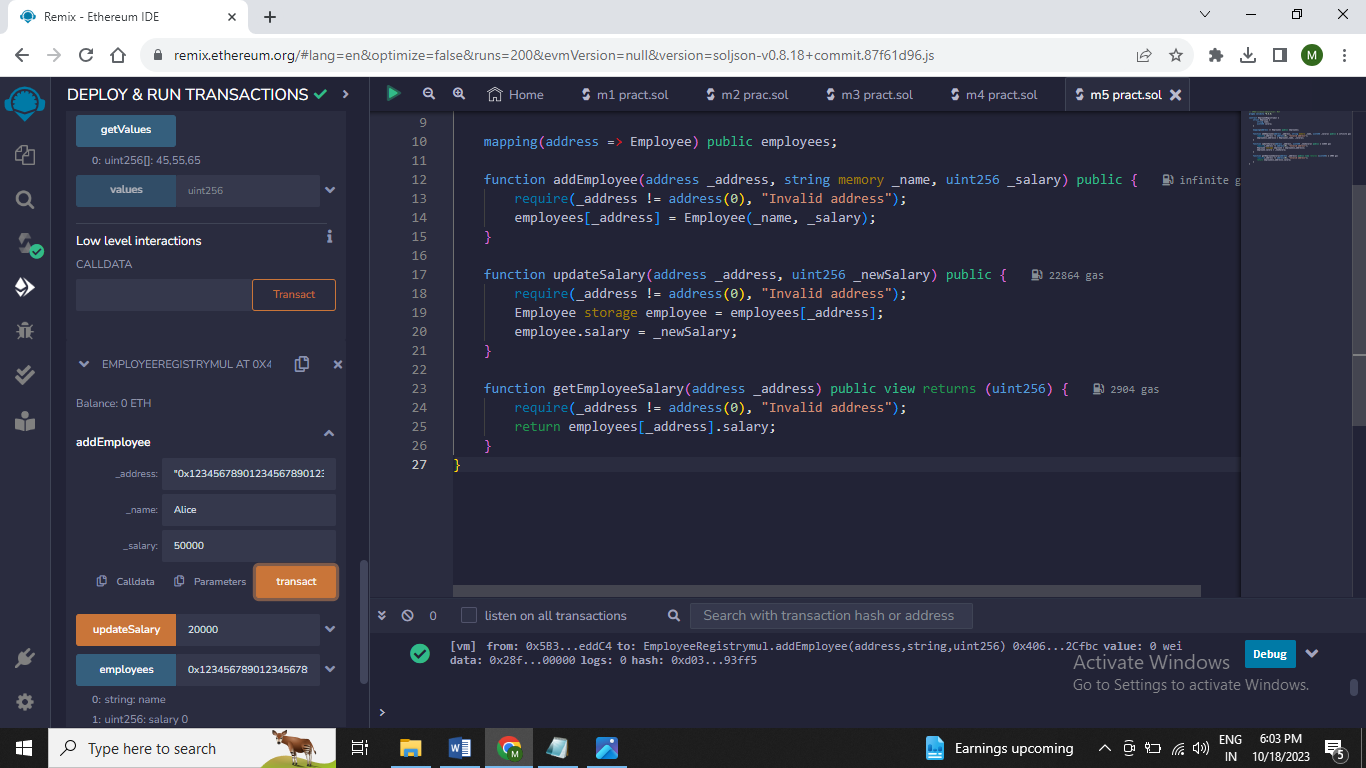
require(\_address != address(0), "Invalid address");

return employees[\_address].salary;

}

}

Output :

****

**Practical 11**

**Aim: Create a Solidity smart contract that utilizes a mapping functionality and deploy it as a contract on the Ethereum blockchain using Remix**

**Code :**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract MappingExample {

mapping(address => uint256) public balances;

constructor() {

balances[msg.sender] = 1000;

}

function setBalance(address \_address, uint256 \_balance) public {

balances[\_address] = \_balance;

}

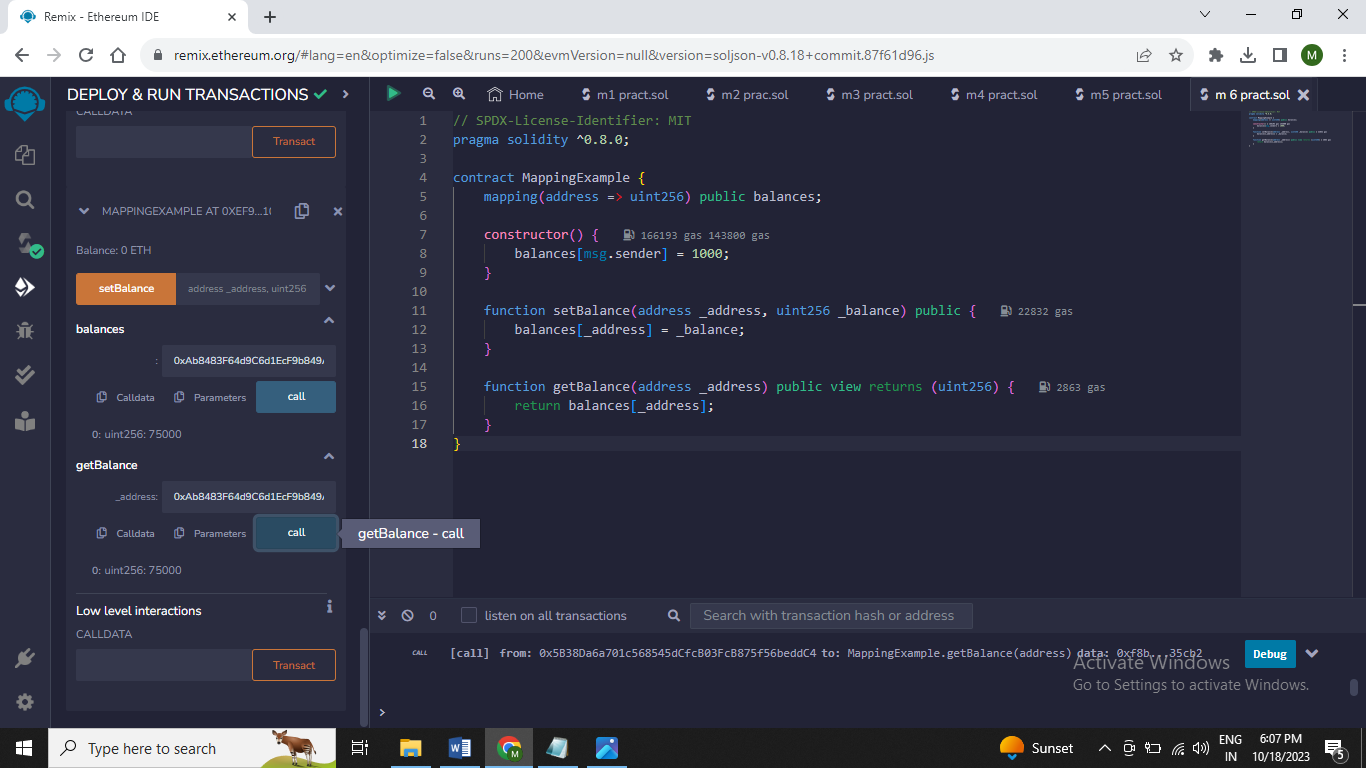
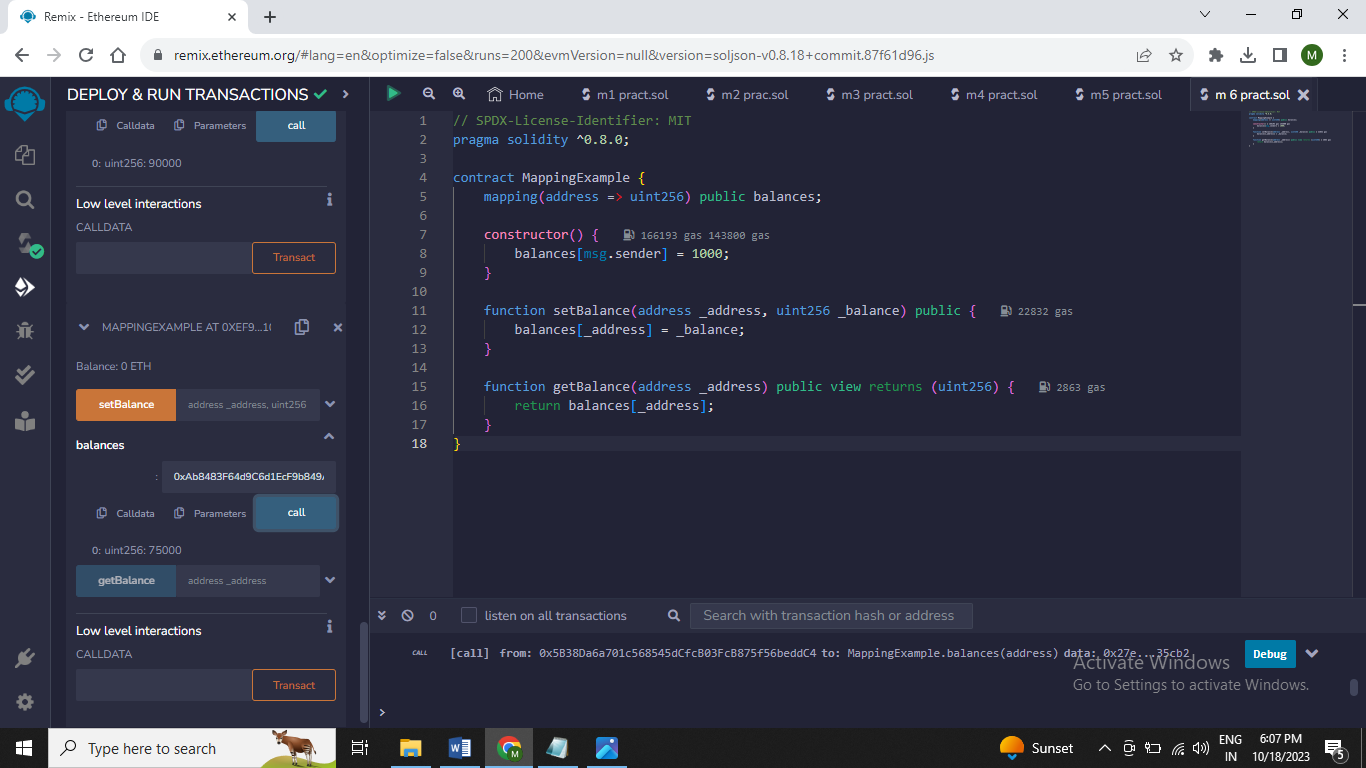
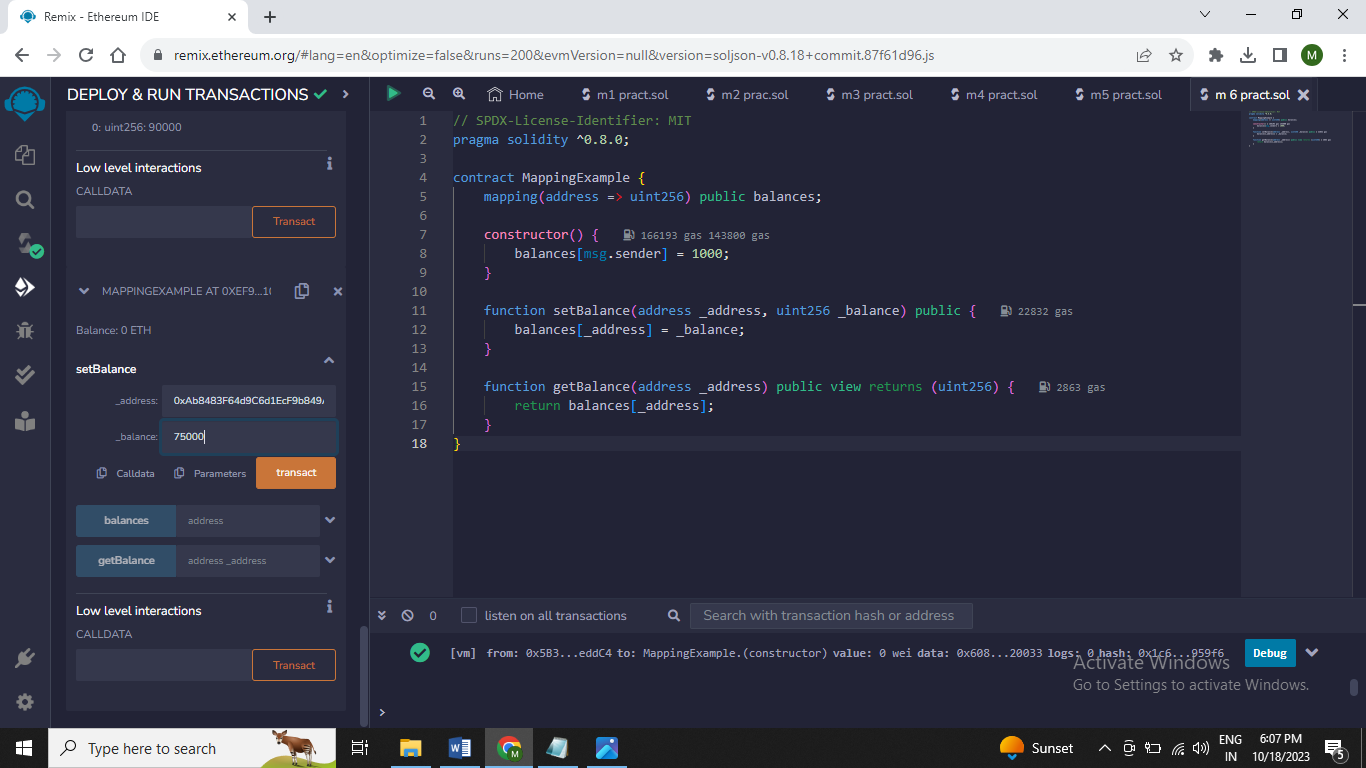
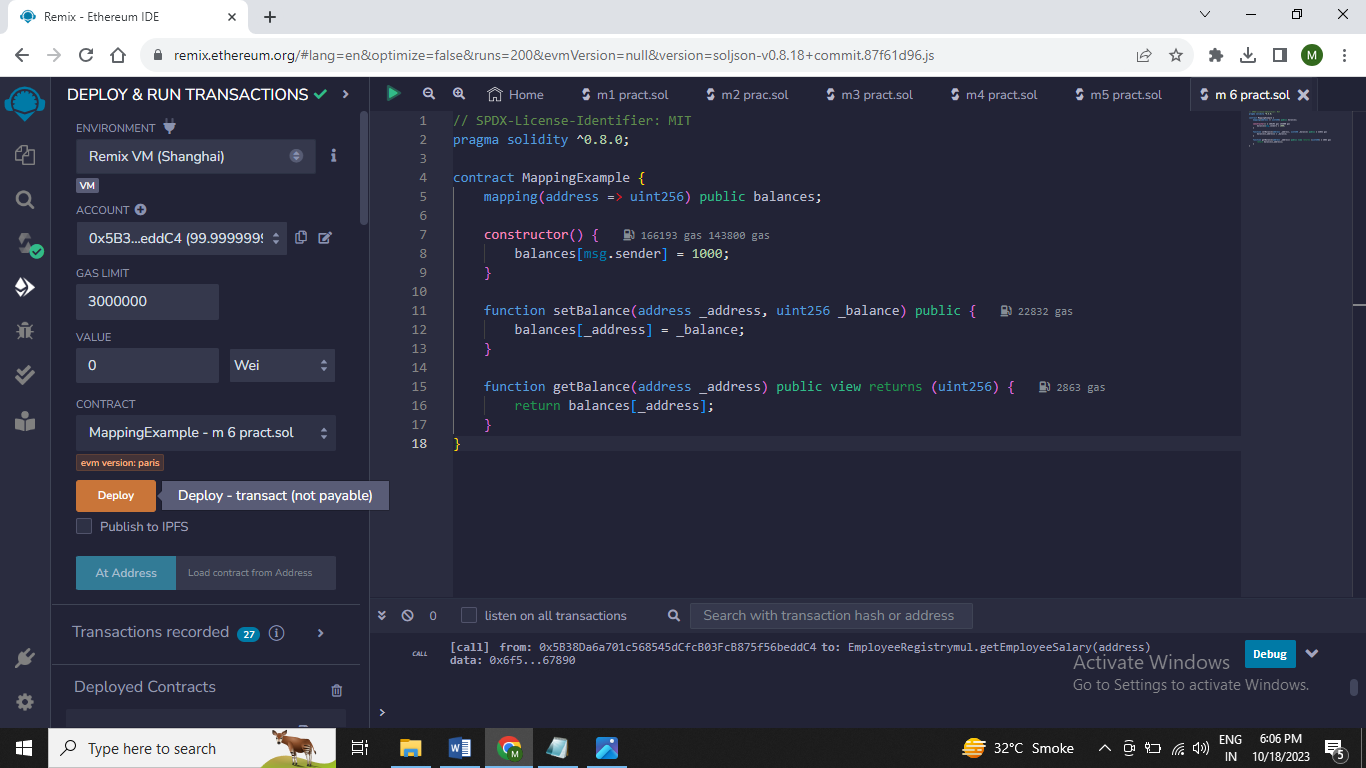
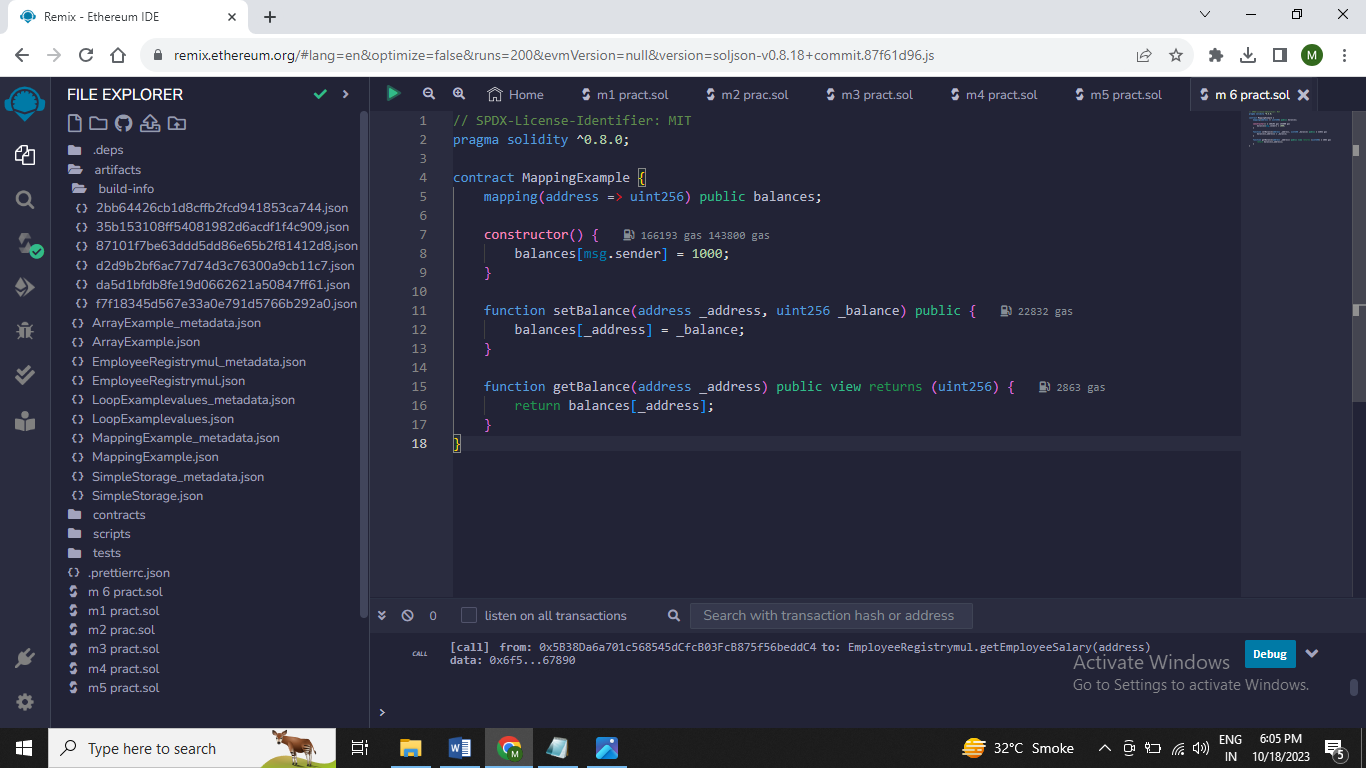
function getBalance(address \_address) public view returns (uint256) {

return balances[\_address];

}

}

Output:

****