Blockchain

Index

Sr. No	Title	Date	Sign
1.	A . simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it. B. Create multiple transactions and display them C. Create a transaction class to send and receive money and test it. D. Create a blockchain, a genesis block and execute it. E. Create a mining function and test it. F. Add blocks to the miner and dump the blockchain.		
2.	write a solidity program for variables, operators, loops, decision making and string.		
3.	A. write a solidity program for string, arrays, enums, structure & mappings. B. write a solidity program for function, view function, pure function & fallback function.		
4.	A. write a solidity program for function overloading, mathematical function & cryptographic functions B. write a solidity program for contract, inheritance, constructors, abstract contracts, interfaces, libraries, assembly, events, error handling		
5.	Install hyperledger-Irhoa		
6.	Demonstrate the running of the blockchain node.		
7.	Demonstrate the running of the blockchain node.		
8.	Demonstrate the use of Bitcoin Core API.		
9.	Build Dapps with angular[using truffle and ganache cli		

Practical 1

A. A simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.

After Creating Ubuntu VM-> Login -> Open Terminal -> Install below packages

```
sudo apt-get update
sudo apt-get install python3
sudo apt-get install python3-pip
pip3 install Crypto
pip3 install pycrypto
```

Code :

pip3 install Crypto

pip3 install pycrypto

```python
import hashlib
import random
import binascii
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:

sender 30819f300d06092a864886f………

B. Create multiple transactions and display them

Code:

```python
import hashlib
import binascii
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,
```

```python
                '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 = []
A = Client()
B = Client()
t1 = Transaction(
    A,
    B.identity,
    15.0
)
t1.sign_transaction()
display_transaction (t1)
```

Output:

sender: 30819f300d0609.......
-----
recipient: 30819f300d06.....
-----
value: 15.0
-----
time: 2022-04-26 04:00:21.070283

C. Create a transaction class to send and receive money and test it

Code:

```python
# following imports are required by PKI
import hashlib
import binascii
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
```

```python
        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)
```

```python
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:

Transaction # 1
sender: 30819f300d060...
-----
recipient: 30819f300d02a864....
-----
value: 15.0
-----
time: 2022-04-26 04:07:59.162213

Transaction # 2
sender: 30819f300d06092a8.....
-----
recipient: 30819f300d06092a8.....

-----
value: 25.0
-----
time: 2022-04-26 04:07:59.165396

Transaction # 3
sender: 30819f300d06092a8648....
-----
recipient: 30819f300d06092a86488...
-----
value: 200.0
-----
time: 2022-04-26 04:07:59.168579

D. Create a blockchain, a genesis block and execute it.

Code:

```python
import hashlib
import binascii
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()
```

```python
    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']))
```

```python
    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:

Number of blocks in the chain: 1
block # 0
sender: Genesis
-----
recipient: 30819f300d06092.....
-----
value: 500.0
-----
time: 2022-04-26 04:24:05.232662

E. Create a mining function and test it.

Code:

```python
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'*2=> '11'
  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",2)
```

Output:

prefix 11
testing=>ab7d1f2b4ba63486a274d7a8c5e4dde793c2d47069ae19ab832dc1177622a182
testing=>cf0a36c4f0c3107cba7a8ebe690db004a01f659bc0aed3b327f01fab0065bf41

testing=>fb0eac040f5f40cd4a39373ca0e6165c07a36db3df510b4c0ad4d45654caeabb
testing=>a298e97de6df74e3856aabbd5aeed9807652d98a9911a6431bdb3bad0ad2a7bd
testing=>7ff8aa3e5b40e1b5bed59ab464c9b98ceff64b2445cc446cc89ecd93330cba1e
....
testing=>1cddb5b7e9af6eda960e734606c33f0ce676a7e557a22ba4d7b9af557b0c0360
testing=>29d2f56130e7b276b3cfb94687ff3b1d5c79b6dc8238fe259aae1f5af19fd8b2
testing=>3a5f4dcfed5301f36be80fd7d42573b1585ea4ef9037e96853affe66d68f8a04
testing=>ddb4d9dc8c7f20443eedc9ac798aebb2c080cc46926dc0151760e37097bf2dcf
testing=>4fb1010880723ce012526941ae6236260852c8e995583d0d2f65b6f9ff655c61
testing=>11038c5fc4f90108f4198097c76c9af5d38c92b48fe27968eacbd89324fe9d2a
after 21 iterations found nonce:
11038c5fc4f90108f4198097c76c9af5d38c92b48fe27968eacbd89324fe9d2a
21

F. Add blocks to the miner and dump the blockchain.

```python
# following imports are required by PKI
import hashlib
import random
import binascii
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):
```

```python
        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):
```

```python
        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


A = Client()
B =Client()
C =Client()
t0 = Transaction (
    "Genesis",
    A.identity,
    500.0
)

t1 = Transaction (
```

```python
    A,
    B.identity,
    40.0
)
t2 = Transaction (
    A,
    C.identity,
    70.0
)
t3 = Transaction (
    B,
    C.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:

```
Number of blocks in the chain: 3
block # 0
sender: Genesis
-----
recipient: 30819f300d0609.....
-----
value: 500.0
-----
time: 2022-04-26 04:30:59.070952
-----
-------------
===================================
block # 1
sender: 30819f300d06092a86.....
-----
recipient: 30819f300d06092a.....
-----
value: 40.0
-----
time: 2022-04-26 04:30:59.071076
-----
-------------
sender: 30819f300d06092a86....
-----
recipient: 30819f300d06092a....
-----
value: 70.0
-----
time: 2022-04-26 04:30:59.071174
-----
-------------
===================================
block # 2
sender: 30819f300d06092a....
-----
```

recipient: 30819f300d06092a....

-----

value: 700.0

-----

time: 2022-04-26 04:30:59.071272

===================================

Practical 2

AIM: write a solidity program for variables, operators, loops, decision making and string.

 A)Variables:

 supports three types of variables.

State Variables – Variables whose values are permanently stored in a contract storage.

Local Variables – Variables whose values are present till function is executing.

Global Variables – Special variables exists in the global namespace used to get information about the blockchain.i.e. blockhash(uint blockNumber) returns (bytes32), block.coinbase (address payable), block.difficulty (uint).....and many more

Step 1: Open this website

https://remix.ethereum.org/

Step 2: Create new file – practical.sol
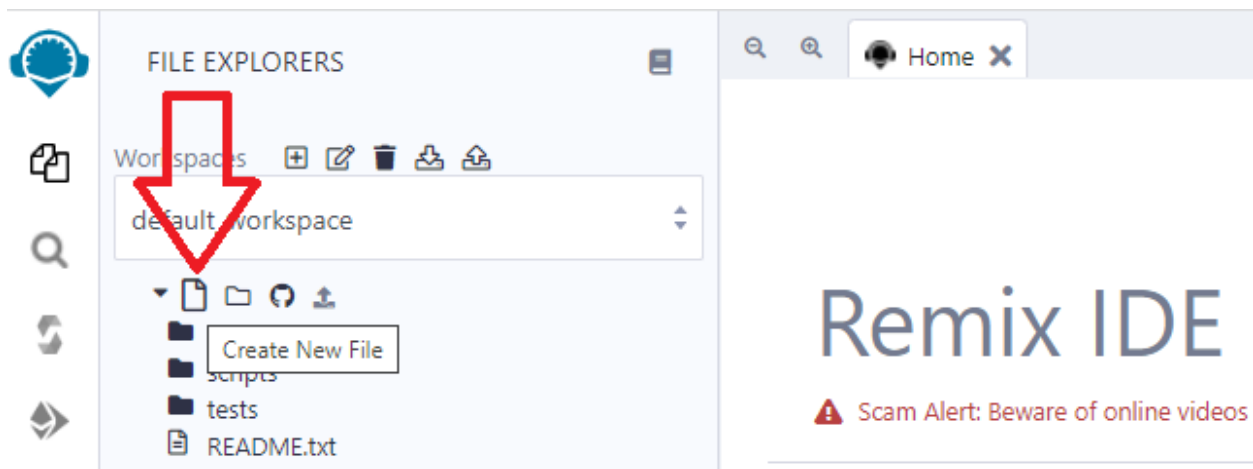
Step 3: Write this program in the new file
//////////////
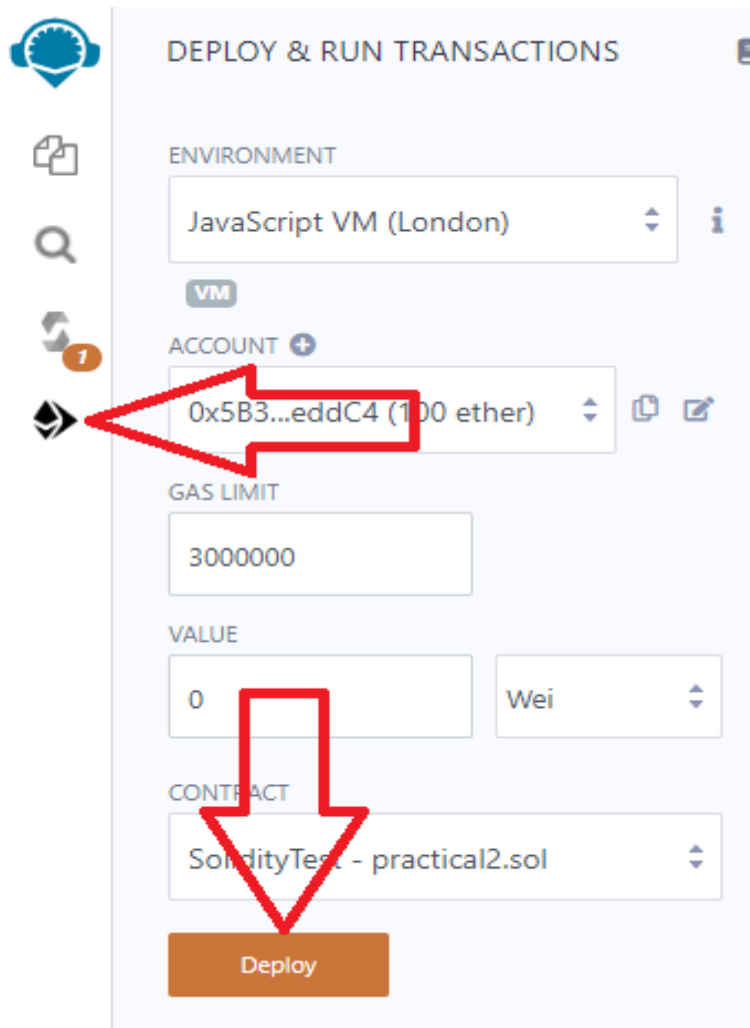
```solidity
pragma solidity ^0.5.0;
contract SolidityTest {
uint storedData; // State variable
constructor() public {
storedData = 10;
}
function getResult() public view returns(uint){
uint a = 1; // local variable
uint b = 2;
uint result = a + b;
return result; //access the state variable
}
}
```

Step 4: Compile contract

Step 5: Deploy contract

Step 6: Select the contract and click button

1.State Variable:

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

Transactions recorded **3**

Deployed Contracts

SOLIDITY_VAR_TEST AT 0XF8E...9FBE8 (I

state_var

0: uint8: 16
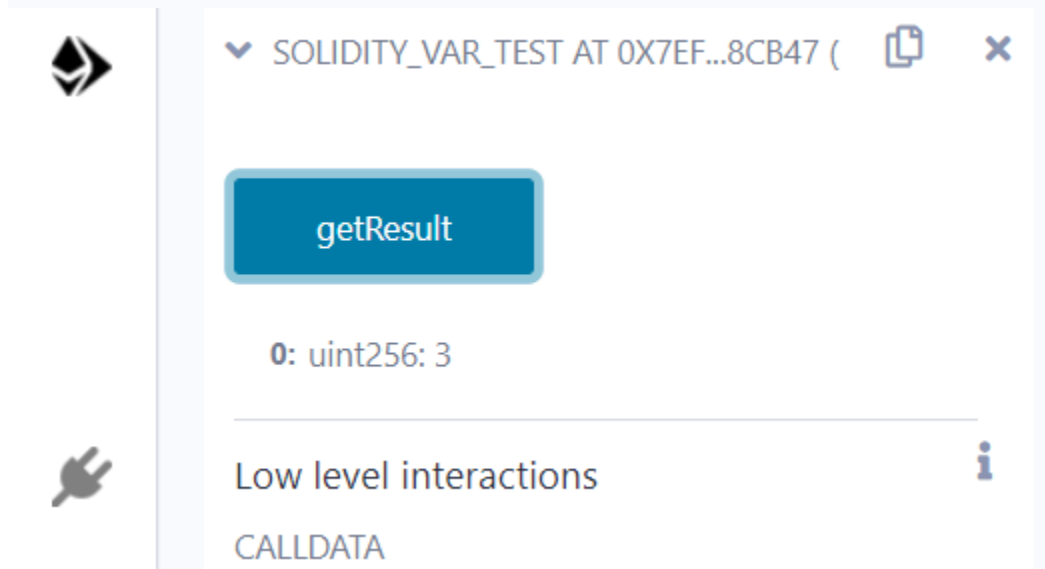
2.Local Variable:

```solidity
// Solidity program to demonstrate
// local variables
pragma solidity ^0.5.0;
```

```solidity
// Creating a contract
contract Solidity_var_Test {
// Defining function to show the declaration and
// scope of local variables
function getResult() public view returns(uint){
// Initializing local variables
uint local_var1 = 1;
uint local_var2 = 2;
uint result = local_var1 + local_var2;
// Access the local variable
return result;
}
}
```

SOLIDITY_VAR_TEST AT 0X7EF...8CB47 (

getResult

0: uint256: 3

Low level interactions

CALLDATA

3.Global variable:

```solidity
// Solidity program to
// show Global variables
pragma solidity ^0.5.0;
// Creating a contract
contract Test {
// Defining a variable
address public admin;
```

```
// Creating a constructor to
// use Global variable
constructor() public {
admin = msg.sender;
}
}
```


Scope of local variables is limited to function in which they are defined but State variables can have three types of scopes.

Public – Public state variables can be accessed internally as well as via messages. For a public state variable, an automatic getter function is generated.

Internal – Internal state variables can be accessed only internally from the current contract or contract deriving from it without using this.

Private – Private state variables can be accessed only internally from the current contract they are defined not in the derived contract from it.

B)Operators

Solidity supports the following types of operators.

Arithmetic Operators

Comparison Operators

Logical (or Relational) Operators

Assignment Operators

Conditional (or ternary) Operators

1. Arithematic Operator

// Solidity contract to demonstrate

// Arithematic Operator

```solidity
pragma solidity ^0.5.0;
// Creating a contract
contract SolidityTest {
// Initializing variables
uint16 public a = 20;
uint16 public b = 10;
// Initializing a variable
// with sum
uint public sum = a + b;
// Initializing a variable
// with the difference
uint public diff = a - b;
// Initializing a variable
// with product
uint public mul = a * b;
// Initializing a variable
// with quotient
uint public div = a / b;
// Initializing a variable
// with modulus
uint public mod = a % b;
// Initializing a variable
// decrement value
uint public dec = --b;
// Initializing a variable
```

```
// with increment value
uint public inc = ++a;
}
```


2.Relational Operator

// Solidity program to demonstrate

// Relational Operator

pragma solidity ^0.5.0;

// Creating a contract
contract SolidityTest {

```solidity
// Declaring variables
uint16 public a = 20;
uint16 public b = 10;

// Initializing a variable
// with bool equal result
bool public eq = a == b;

// Initializing a variable
// with bool not equal result
bool public noteq = a != b;

// Initializing a variable
// with bool greater than result
bool public gtr = a > b;

// Initializing a variable
// with bool less than result
bool public les = a < b;

// Initializing a variable
// with bool greater than equal to result
bool public gtreq = a >= b;

// Initializing a variable
// bool less than equal to result
bool public leseq = a <= b;
}
```

3.Logical Operators

```solidity
// Solidity program to demonstrate

// Logical Operators

pragma solidity ^0.5.0;

// Creating a contract
contract logicalOperator{
```

```solidity
// Defining function to demonstrate
// Logical operator
function Logic(
bool a, bool b) public view returns(
bool, bool, bool){

// Logical AND operator
bool and = a&&b;

// Logical OR operator
bool or = a||b;

// Logical NOT operator
bool not = !a;
return (and, or, not);
}
}
```

4.Bitwise Operators

// Solidity program to demonstrate

// Bitwise Operator

```solidity
pragma solidity ^0.5.0;

// Creating a contract
contract SolidityTest {

// Declaring variables
uint16 public a = 20;
uint16 public b = 10;

// Initializing a variable
// to '&' value
uint16 public and = a & b;

// Initializing a variable
```

```solidity
// to '|' value
uint16 public or = a | b;

// Initializing a variable
// to '^' value
uint16 public xor = a ^ b;

// Initializing a variable
// to '<<' value
uint16 public leftshift = a << b;

// Initializing a variable
// to '>>' value
uint16 public rightshift = a >> b;

// Initializing a variable
// to '~' value
uint16 public not = ~a ;


}
```

5.Assignment Operator

```solidity
// Solidity program to demonstrate

// Assignment Operator

pragma solidity ^0.5.0;

// Creating a contract
contract SolidityTest {

// Declaring variables
uint16 public assignment = 20;
uint public assignment_add = 50;
uint public assign_sub = 50;
```

```solidity
uint public assign_mul = 10;
uint public assign_div = 50;
uint public assign_mod = 32;

// Defining function to
// demonstrate Assignment Operator
function getResult() public{
assignment_add += 10;
assign_sub -= 20;
assign_mul *= 10;
assign_div /= 10;
assign_mod %= 20;
return ;
}
}
```

6.**Conditional Operators**

// Solidity program to demonstrate

// Conditional Operator

```solidity
pragma solidity ^0.5.0;

// Creating a contract
contract SolidityTest{
// Defining function to demonstrate
// conditional operator
function sub(
uint a, uint b) public view returns(
uint){
uint result = (a > b? a-b : b-a);
return result;
}
}
```

C)Loops:

1.While loop: The most basic loop in Solidity is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

2.do-while loop: The **do…while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

3.for loop: The **for** loop is the most compact form of looping. It includes the following three important parts –

The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.

The **iteration statement** where you can increase or decrease your counter.

4.loop control: Solidity provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.To handle all such situations, Solidity provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

1.While Loop

```solidity
pragma solidity ^0.5.0;
contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}
function getResult() public view returns(string memory){
```

```solidity
uint a = 10;
uint b = 2;
uint result = a + b;
return integerToString(result);
}
function integerToString(uint _i) internal pure
returns (string memory) {
if (_i == 0) {
return "0";
}
uint j = _i;
uint len;
while (j != 0) {
len++;
j /= 10;
}
bytes memory bstr = new bytes(len);
uint k = len - 1;
while (_i != 0) { // while loop
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
}
return string(bstr);
}
}
```

2.Do-while loop:

```solidity
pragma solidity ^0.5.0;
contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}
function getResult() public view returns(string memory){
uint a = 10;
uint b = 2;
uint result = a + b;
```

```solidity
    return integerToString(result);
}
function integerToString(uint _i) internal pure
returns (string memory) {
if (_i == 0) {
return "0";
}
uint j = _i;
uint len;
while (j != 0) {
len++;
j /= 10;
}
bytes memory bstr = new bytes(len);
uint k = len - 1;
do {            // do while loop
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
}
while (_i != 0);
return string(bstr);
}
}
```

3.For Loop:

```solidity
pragma solidity ^0.5.0;

contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}

function getResult() public view returns(string memory){
uint a = 10;
uint b = 2;
uint result = a + b;
```

```solidity
return integerToString(result);
}

function integerToString(uint _i) internal pure
returns (string memory) {
if (_i == 0) {
return "0";
}
uint j=0;
uint len;
for (j = _i; j != 0; j /= 10) {  //for loop example
len++;
}
bytes memory bstr = new bytes(len);
uint k = len - 1;
while (_i != 0) {
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
}
return string(bstr);//access local variable
}}
```

4.loop Control: (Break statement)

```solidity
pragma solidity ^0.5.0;

contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}
function getResult() public view returns(string memory){
uint a = 1;
uint b = 2;
uint result = a + b;
return integerToString(result);
}
function integerToString(uint _i) internal pure
returns (string memory) {
```

```solidity
if (_i == 0) {
return "0";
}
uint j = _i;
uint len;

while (true) {
len++;
j /= 10;
if(j==0){
break;  //using break statement
}
}
bytes memory bstr = new bytes(len);
uint k = len - 1;

while (_i != 0) {
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
}
return string(bstr);
}
}
```

(continue statement)

```solidity
pragma solidity ^0.5.0;
contract SolidityTest {
uint storedData;
constructor() public{
storedData = 10;
}
function getResult() public view returns(string memory){
uint n = 1;
uint sum = 0;

while( n < 10){
n++;
if(n == 5){
continue; // skip n in sum when it is 5.
```

```solidity
}
sum = sum + n;
}
return integerToString(sum);
}
function integerToString(uint _i) internal pure
returns (string memory) {

if (_i == 0) {
return "0";
}
uint j = _i;
uint len;

while (true) {
len++;
j /= 10;
if(j==0){
break;   //using break statement
}
}
bytes memory bstr = new bytes(len);
uint k = len - 1;

while (_i != 0) {
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
}
return string(bstr);
}
}
```

D) Decision Making:

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.Solidity supports conditional

statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.

1.if statement: The **if** statement is the fundamental control statement that allows Solidity to make decisions and execute statements conditionally.

```solidity
pragma solidity ^0.5.0;

contract SolidityTest {
uint storedData;
constructor() public {
storedData = 10;
}
function getResult() public view returns(string memory){
uint a = 1;
uint b = 2;
uint result = a + b;
return integerToString(result);
}
function integerToString(uint _i) internal pure
returns (string memory) {
if (_i == 0) {   // if statement
return "0";
}
uint j = _i;
uint len;

while (j != 0) {
len++;
j /= 10;
}
bytes memory bstr = new bytes(len);
uint k = len - 1;

while (_i != 0) {
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
}
return string(bstr);//access local variable
}}
```

2.if-else statement: The **'if...else'** statement is the next form of control statement that allows Solidity to execute statements in a more controlled way.

```solidity
pragma solidity ^0.5.0;

// Creating a contract
contract Types {
// Declaring state variables
uint i = 10;
bool even;

// Defining function to
// demonstrate the use of
// 'if...else statement'
function decision_making(
) public payable returns(bool){
if (i%2 == 0){
even = true;
}
else{
even = false;
}
return even;
}
}
```

3.if-else..if statement: The **if...else if...** statement is an advanced form of **if...else** that allows Solidity to make a correct decision out of several conditions.

```solidity
pragma solidity ^0.5.0;

// Creating a contract
contract Types {
// Declaring state variables
uint i = 12;
string result;
// Defining function to
```

```solidity
// demonstrate the use
// of 'if...else if...else
// statement'
function decision_making (
) public returns(string memory){
if(i<10){
result = "less than 10";
}
else if(i == 10){
result = "equal to 10";
}
else{
result = "greater than 10";
}
return result;
}
}
```

String:

```solidity
// Solidity program to demonstrate

// how to create a contract

pragma solidity ^0.4.23;

// Creating a contract
contract Test {
// Declaring variable
string  str;

// Defining a constructor
constructor(string str_in){
str = str_in;
}
// Defining a function to
// return value of variable 'str'
function str_out() public view returns(string memory){
return str;
}
}
```

Practical 3

AIM: write a solidity program for variables, operators, loops, decision making and string.

A) String:

Solidity supports String literal using both double quote (") and single quote ('). It provides string as a data type to declare a variable of type String.(Int to str)

```solidity
pragma solidity ^0.5.0;

contract SolidityTest {
constructor() public{
}
function getResult() public view returns(string memory){
uint a = 1;
uint b = 2;
uint result = a + b;
return integerToString(result);
}
function integerToString(uint _i) internal pure
returns (string memory) {

if (_i == 0) {
return "0";
}
uint j = _i;
uint len;

while (j != 0) {
len++;
j /= 10;
}
bytes memory bstr = new bytes(len);
uint k = len - 1;

while (_i != 0) {
bstr[k--] = byte(uint8(48 + _i % 10));
```

```
_i /= 10;
}
return string(bstr);
}
}
```

B)Array:

Array is a data structure, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

```
// Solidity program to demonstrate
// accessing elements of an array
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

// Declaring an array
uint[6] data;
uint x;

// Defining function to
// assign values to array
function array_example() public returns (uint[6] memory)
{

data = [uint(10), 20, 30, 40, 50, 60];
}
function result() public view returns(uint[6] memory){
return data;
}
// Defining function to access
// values from the array
// from a specific index
function array_element() public view returns (uint){
uint x = data[2];
return x;
}
```

```
}
```

C)Enums:

Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums. With the use of enums it is possible to reduce the number of bugs in your code.

```solidity
// Solidity program to demonstrate
// how to use 'enumerator'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

// Creating an enumerator
enum week_days
{
Monday,
Tuesday,
Wednesday,
Thursday,
Friday,
Saturday,
Sunday
}
// Declaring variables of
// type enumerator
week_days week;

week_days choice;

// Setting a default value
week_days constant default_value
= week_days.Sunday;

// Defining a function to
// set value of choice
function set_value() public {
```

```solidity
choice = week_days.Thursday;
}
// Defining a function to
// return value of choice
function get_choice(
) public view returns (week_days) {
return choice;
}
// Defining function to
// return default value
function getdefaultvalue(
) public pure returns(week_days) {
return default_value;
}
}
```

D)Structure:

Struct types are used to represent a record.

```solidity
pragma solidity ^0.5.0;

contract test {
struct Book {
string title;
string author;
uint book_id;
}
Book book;

function setBook() public {
book = Book('Learn Java', 'TP', 1);
}
function getBookId() public view returns (uint) {
return book.book_id;
}
}
```

E)Mappings:

Mapping is a reference type as arrays and structs. Following is the syntax to declare a mapping type.

mapping(_KeyType => _ValueType) where ,

_KeyType – can be any built-in types plus bytes and string. No reference type or complex objects are allowed.

_ValueType – can be any type.

```solidity
pragma solidity ^0.5.0;

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

function updateBalance() public  returns(uint) {
balance[msg.sender]=30;
return balance[msg.sender];
}
}
```

Mapping program for String.

```solidity
pragma solidity ^0.5.0;

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

function updateBalance() public returns(string memory){
name[msg.sender] = "Mrunali";
return name[msg.sender];
}
function printsender() public view returns(address) {
return msg.sender;
}
}
```

3B. WRITE A SOLIDITY PROGRAM FOR FUNCTION, VIEW FUNCTION, PURE

FUNCTION & FALLBACK FUNCTION.

A)Function:

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

```solidity
pragma solidity ^0.5.0;

contract SolidityTest {
constructor() public{
}
function getResult() public view returns(string memory){
uint a = 1;
uint b = 2;
uint result = a + b;
return integerToString(result);
}
function integerToString(uint _i) internal pure
returns (string memory) {

if (_i == 0) {
return "0";
}
uint j = _i;
uint len;

while (j != 0) {
len++;
j /= 10;
}
bytes memory bstr = new bytes(len);
uint k = len - 1;

while (_i != 0) {
bstr[k--] = byte(uint8(48 + _i % 10));
_i /= 10;
```

```
}
return string(bstr);//access local variable


}
}
```

B)View Function:

View functions ensure that they will not modify the state. A function can be declared as **view**. Getter method are by default view functions.

```
pragma solidity ^0.5.0;
contract Test {
function getResult() public view returns(uint product, uint sum){
uint a = 1; // local variable
uint b = 2;
product = a * b;
sum = a + b;
}
}
```

C)Pure Function:

Pure functions ensure that they not read or modify the state. A function can be declared as **pure**. Pure functions can use the revert() and require() functions to revert potential state changes if an error occurs.

```
pragma solidity ^0.5.0;

contract Test {
function getResult() public pure returns(uint product, uint sum){
uint a = 1;
uint b = 2;
product = a * b;
sum = a + b;
}
}
```

D)Fallback Function:

Fallback function is a special function available to a contract.

```solidity
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));
}
}
```

Practical 4

A. write a solidity program for function overloading, mathematical function & cryptographic functions

Function Overloading:

The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

```solidity
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(2,2);
}
function callSumWithThreeArguments() public pure returns(uint){
return getSum(1,2,4);
}

}
```

Mathematical Function:

Solidity provides inbuilt mathematical functions as well.

```solidity
pragma solidity ^0.5.0;

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

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

Cryptographic Function:

Solidity provides inbuilt cryptographic functions as well.

```solidity
pragma solidity ^0.5.0;
contract Test {
function callKeccak256() public pure returns(bytes32 result){
return keccak256("ABC");
}
}
```

4B. WRITE A SOLIDITY PROGRAM FOR CONTRACT, INHERITANCE, CONSTRUCTORS, ABSTRACT CONTRACTS, INTERFACES, LIBRARIES, ASSEMBLY, EVENTS, ERROR HANDLING.

A)Contract:

Contract in Solidity is similar to a Class in C++. A Contract have following properties.

Constructor – A special function declared with constructor keyword which will be executed once per contract and is invoked when a contract is created.

State Variables – Variables per Contract to store the state of the contract.

Functions – Functions per Contract which can modify the state variables to alter the state of a contract.

// Calling function from external contract

```solidity
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;
```

```solidity
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(); }
}
```

B)Inheritance:

Inheritance is a way to extend functionality of a contract. Solidity supports both single as well as multiple inheritance.

```solidity
// Solidity program to

// demonstrate

// Single Inheritance

pragma solidity >=0.4.22 <0.6.0;

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

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

```solidity
contract child is parent{

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

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

// Defining function to call
// setValue and getValue functions
function testInheritance() public {
cc.setValue();
}
function result() public view returns(uint ){
return cc.getValue();
}
}
```

C)Constructors:

Constructor is a special function declared using constructor keyword. It is an optional function and is used to initialize state variables of a contract. Following are the key characteristics of a constructor.

A contract can have only one constructor.

A constructor code is executed once when a contract is created and it is used to initialize contract state.

A constructor can be either public or internal.

An internal constructor marks the contract as abstract.

In case, no constructor is defined, a default constructor is present in the contract.

```solidity
pragma solidity ^0.5.0;
contract Base {
uint data;
constructor(uint _data) public {
data = _data;
}
function getresult()public view returns(uint){
return data;
}
}
contract Derived is Base (5) {
constructor() public {}
}
```

// Indirect Initialization of Base Constructor

```solidity
pragma solidity ^0.5.0;

contract Base {
uint data;
constructor(uint _data) public {
data = _data;
}
function getresult()public view returns(uint){
return data;
}
}
contract Derived is Base {
constructor(uint _info) Base(_info * _info) public {}
}
```

D)Abstract Contracts:

Abstract Contract is one which contains at least one function without any implementation. Such a contract is used as a base contract. Generally an abstract contract contains both implemented as well as abstract functions. Derived contract will implement the abstract function and use the existing functions as and when required.

```solidity
pragma solidity ^0.5.0;

contract Calculator {
function getResult() public view returns(uint);
}
contract Test is Calculator {
function getResult() public view returns(uint) {
uint a = 4;
uint b = 2;
uint result = a + b;
return result;
}
}
```

E)Interfaces:

Interfaces are similar to abstract contracts and are created using interface keyword. Following are the key characteristics of an interface.

Interface can not have any function with implementation.

Functions of an interface can be only of type external.

Interface can not have constructor.

Interface can not have state variables.

```solidity
pragma solidity ^0.5.0;

interface Calculator {
function getResult() external view returns(uint);
}
contract Test is Calculator {
constructor() public {}
```

```solidity
function getResult() external view returns(uint){
uint a = 5;
uint b = 2;
uint result = a + b;
return result;
}
}
```

F)Libraries:

Libraries are similar to Contracts but are mainly intended for reuse. A Library contains functions which other contracts can call. Solidity have certain restrictions on use of a Library.

```solidity
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;
uint value;
uint index;
constructor() public {
data.push(6);
data.push(7);
data.push(8);
data.push(9);
data.push(10);
}
function isValuePresent() external {
value = 9;
//search if value is present in the array using Library function
index = Search.indexOf(data, value);
}
```

```solidity
function getresult() public view returns(uint){
return index;
}}
```

G)Assembly:

Solidity provides an option to use assembly language to write inline assembly within Solidity source code. We can also write a standalone assembly code which then be converted to bytecode. Standalone Assembly is an intermediate language for a Solidity compiler and it converts the Solidity code into a Standalone Assembly and then to byte code. We can used the same language used in Inline Assembly to write code in a Standalone assembly.

```solidity
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);
}
}
```

H)Events:

Event is an inheritable member of a contract. An event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain. An event generated is not accessible from within contracts, not even the one which have created and emitted them.

// Solidity program to demonstrate

// creating an event

```solidity
pragma solidity ^0.4.21;

// Creating a contract
contract eventExample {

// Declaring state variables
uint256 public value = 0;

// Declaring an event
event Increment(address owner);

// Defining a function for logging event
function getValue(uint _a, uint _b) public {
emit Increment(msg.sender);
value = _a + _b;
}
}
```

I)Error Handling:

Solidity provides various functions for error handling. Generally when an error occurs, the state is reverted back to its original state. Other checks are to prevent unauthorized code access.

Solidity program to demonstrate require statement.

// Solidity program to

// demonstrate require

// statement

```solidity
pragma solidity ^0.5.0;
// Creating a contract
contract requireStatement {
// Defining function to
// check input
function checkInput(uint8 _input) public view returns(string memory){
require(_input >= 0, "invalid uint");
require(_input <= 255, "invalid uint8");

return "Input is Uint8";
}
// Defining function to
// use require statement
function Odd(uint _input) public view returns(bool){
require(_input % 2 != 0);
return true;
}
}
```

Solidity program to demonstrate assert statement.

// Solidity program to

// demonstrate assert

// statement

```solidity
pragma solidity ^0.5.0;

// Creating a contract
contract assertStatement {
// Defining a state variable
bool result;
// Defining a function
// to check condition
function checkOverflow(uint8 _num1, uint8 _num2) public {
uint8 sum = _num1 + _num2;
assert(sum<=255);
result = true;
```

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

Solidity program to demonstrate revert statement.

// Solidity program to

// demonstrate revert

```
pragma solidity ^0.5.0;
// Creating a contract
contract revertStatement {
// Defining a function
// to check condition
function checkOverflow(uint _num1, uint _num2) public view returns(
string memory, uint) {
uint sum = _num1 + _num2;
if(sum < 0 || sum > 255){
revert(" Overflow Exist");
}
else{
return ("No Overflow", sum);
}
}
}
```

Practical 5

Aim: Install hyperledger-Irhoa

Step 1: install docker
sudo apt-get install curl

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo apt-get update

apt-cache policy docker-ce

sudo apt-get install -y docker-ce

Step 2: create docker network
sudo docker network create mithilesh-iroha-network

Step 3:add PostgreSQL to our network

sudo docker run --name some-postgres -e POSTGRES_USER=postgres -e
POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 --network=mithilesh-
iroha-network -d postgres:9.5

Step 4:create a volume of persistant storage named "blockstore" to store the blocks for our
blockchain
sudo docker volume create blockstore

Step 5: Download the Iroha code from github.
sudo apt-get install git
git clone -b develop https://github.com/hyperledger/iroha --depth=1

Step 6: run the Iroha docker container

**sudo docker run -it --name iroha **
**-p 50051:50051 **
**-v $(pwd)/iroha/example:/opt/iroha_data **
**-v blockstore:/tmp/block_store **

**--network=mithilesh-iroha-network **
**--entrypoint=/bin/bash **
hyperledger/iroha:latest

Step 7: run Iroha
irohad --config config.docker --genesis_block genesis.block --keypair_name node0

Step 8:Open a new terminal
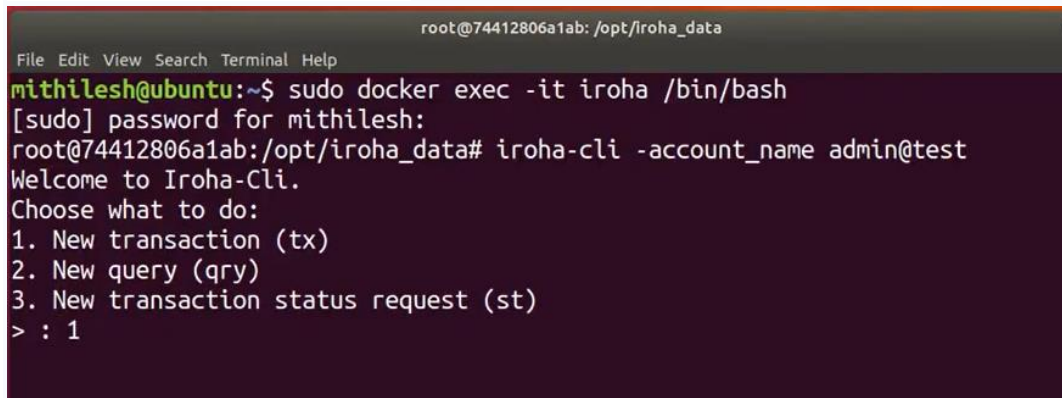
Step 9:attach the docker container to our terminal
sudo docker exec -it iroha /bin/bash

Step 10:Launch the iroha-cli tool and login as admin@test.

iroha-cli -account_name admin@test

Select 1 – for new transaction

Select 14- for creating new coin

File Edit View Search Terminal Help
> : 1
Forming a new transactions, choose command to add:
1. Detach role from account (detach)
2. Add new role to account (apnd_role)
3. Create new role (crt_role)
4. Set account key/value detail (set_acc_kv)
5. Transfer Assets (tran_ast)
6. Grant permission over your account (grant_perm)
7. Subtract Assets Quantity (sub_ast_qty)
8. Set Account Quorum (set_qrm)
9. Remove Signatory (rem_sign)
10. Create Domain (crt_dmn)
11. Revoke permission from account (revoke_perm)
12. Create Account (crt_acc)
13. Add Signatory to Account (add_sign)
14. Create Asset (crt_ast)
15. Add Peer to Iroha Network (add_peer)
16. Add Asset Quantity (add_ast_qty)
0. Back (b)
> : 14

Now type Asset name: mscit
Domain id: test
Asset precision: 2
And select option 3 to add more command

Asset name: mscit
Domain Id: test
Asset precision: 2
Command is formed. Choose what to do:
1. Go back and start a new transaction (b)
2. Save as json file (save)
3. Add one more command to the transaction (add)
4. Send to Iroha peer (send)
> : 3

Now select option 16 to add asset quantity
Asset id: mscit#test
Amount: 16.35
Select option 4- send request to Iroha peer

```
15. Add Peer to Iroha Network (add_peer)
16. Add Asset Quantity (add_ast_qty)
0. Back (b)
> : 16
Asset Id: mscit#test
Amount to add, e.g 123.456: 16.35
Command is formed. Choose what to do:
1. Go back and start a new transaction (b)
2. Save as json file (save)
3. Add one more command to the transaction (add)
4. Send to Iroha peer (send)
> : 4
Peer address (127.0.0.1): 127.0.0.1
Peer port (50051): 50051
```

```
4. Send to Iroha peer (send)
> : 4
Peer address (127.0.0.1): 127.0.0.1
Peer port (50051): 50051
[2022-04-25 14:29:47.336304284][I][CLI/ResponseHandler/Transaction]: Transaction
 successfully sent
Congratulation, your transaction was accepted for processing.
Its hash is 70a37c3b8c32ac6d569c19e47105cc2eb17935218c00c5bff49526962631e6a8
--------------------
```

Select option 2 –for query
Select option 8- for assets

Select option 1
Enter peer : 127.0.0.1
Port: 50051

Aim: Demonstrate the use of Bitcoin Core API.

```python
#pip install bitcoinlib
from bitcoinlib.wallets import Wallet
w = Wallet.create('Wallet1')
key1 = w.get_key()
print(key1.address)
w.scan()
print(w.info())
```

```
*IDLE Shell 3.9.12*                                                          —  □  ×
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.12 (tags/v3.9.12:b28265d, Mar 23 2022, 23:39:15) [MSC v.1929 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:/Users/Mithilesh/Desktop/btc/program1.py ============
1LAHcWggjV6XqoTEmsK6FRNMGkou3Ldh2U
```

More detail on: https://pypi.org/project/bitcoinlib/

Practical 6

Aim: Create your own blockchain and demonstrate its use.

Install on Ubuntu via PPAs

The easiest way to install go-ethereum on Ubuntu-based distributions is with the built-in launchpad PPAs (Personal Package Archives). We provide a single PPA repository that contains both our stable and development releases for Ubuntu versions trusty, xenial, zesty and artful.

linux:

To enable our launchpad repository run:

Step 1: open new terminal

Step 2: on terminal type this command

sudo add-apt-repository -y ppa:ethereum/ethereum

#if above command gives error then run

#sudo apt-get install --reinstall ca-certificates

Step 3: install the stable version of go-ethereum:

sudo apt-get update

sudo apt-get install ethereum

Step 4: create new directory for storing blockchain data

mkdir myblockchain

cd myblockchain

Step 5:Create genesis.json file

sudo nano genesis.json

```json
{
  "config": {
    "chainId": 10,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "eip150Block": 0,
    "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000"
  },
  "alloc": {},
  "coinbase": "0x0000000000000000000000000000000000000000",
  "difficulty": "0x02000000",
  "extraData": "",
  "gasLimit": "0x2fefd8",
  "nonce": "0x0000000000000042",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp": "0x00"
}
```

save the file -> ctrl +o to write -> {enter} save -> ctrl +x exit

Step 6: initialize the block

sudo geth --datadir TestChain init genesis.json

Step 7: create network

sudo geth –datadir TestChain –networkid 1234

[do not close this terminal]

///

Step 8: open new terminal 2:

cd myblockchain

Step 9: attach geth to the network

sudo geth attach TestChain/geth.ipc

Step 10: on geth terminal type these commands

personal.newAccount("123456")

miner.start()

miner.setEtherbase(eth.accounts[0])

admin.addPeer(admin.nodeInfo.enode)

Step 10: Wait for 10-20 minutes and check balance

eth.getBalance(eth.accounts[0])

if ether balance is 0 wait for 10-20minutes for mining process to get complete and run **eth.getBalance(eth.accounts[0]) again.**

After balance is updated you can check current block height

eth.blockNumber

Practical 7

Aim: Demonstrate the running of the blockchain node.

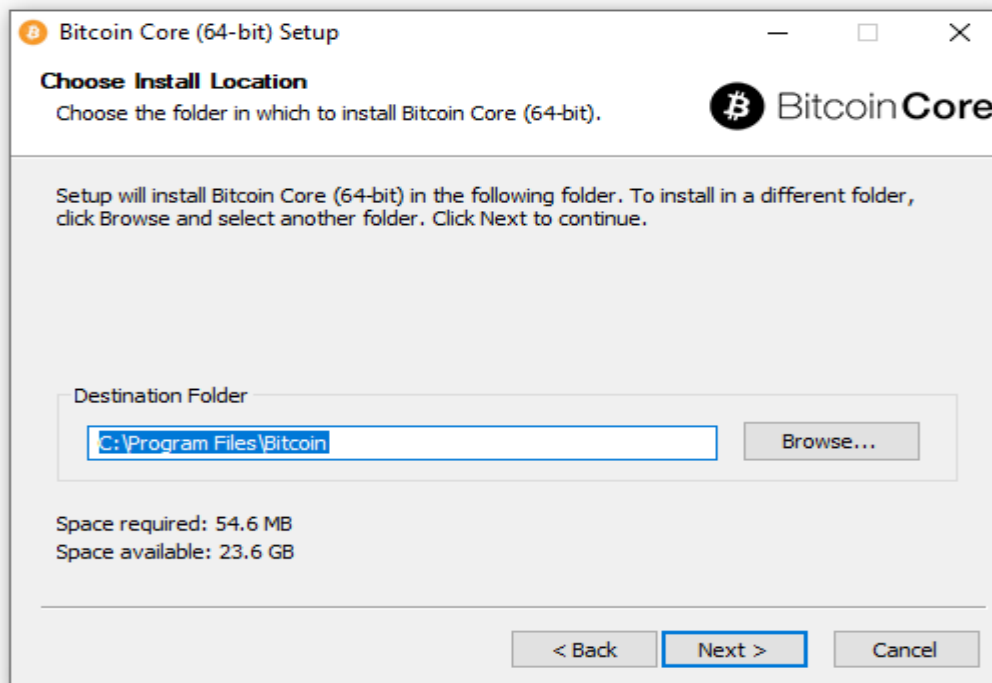Step 1: Visit: https://bitcoin.org/en/download

Step 2: Download windows setup [use and try with Linux version as well]
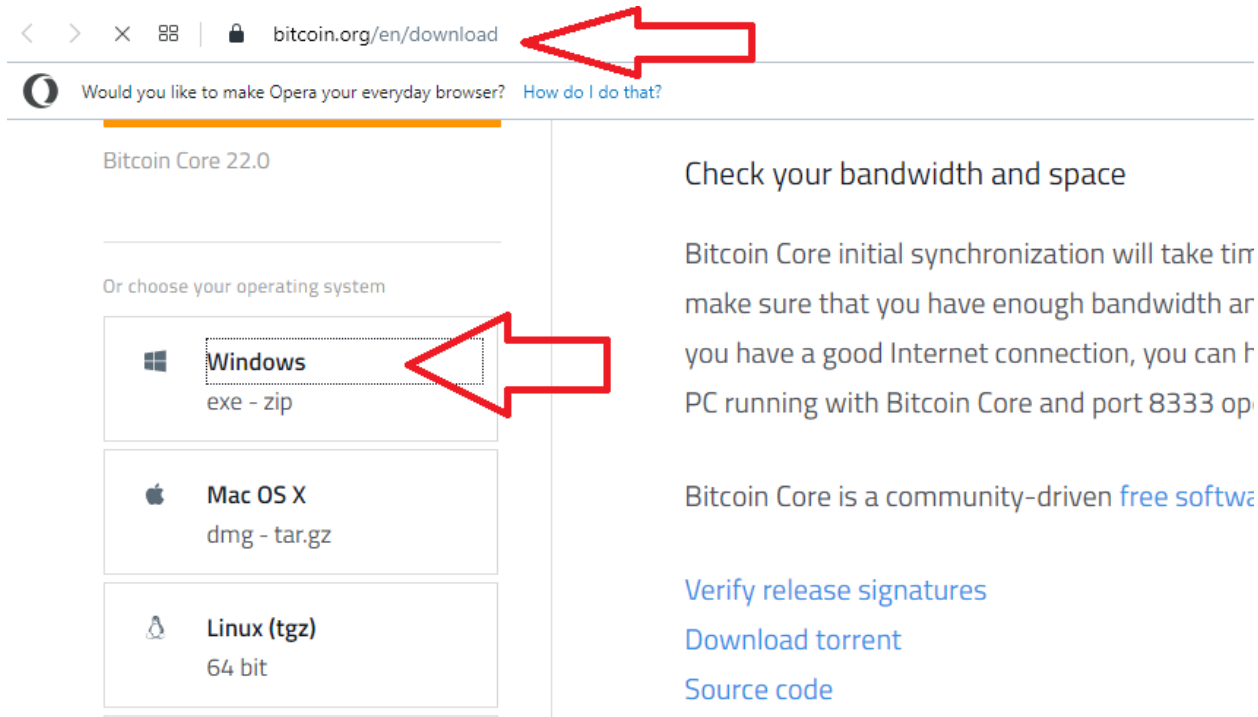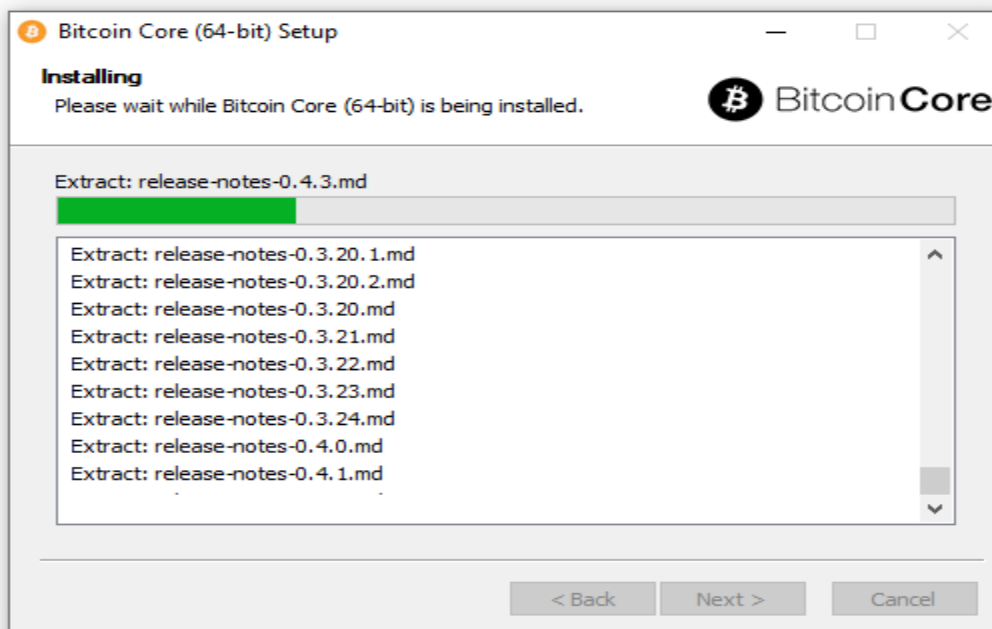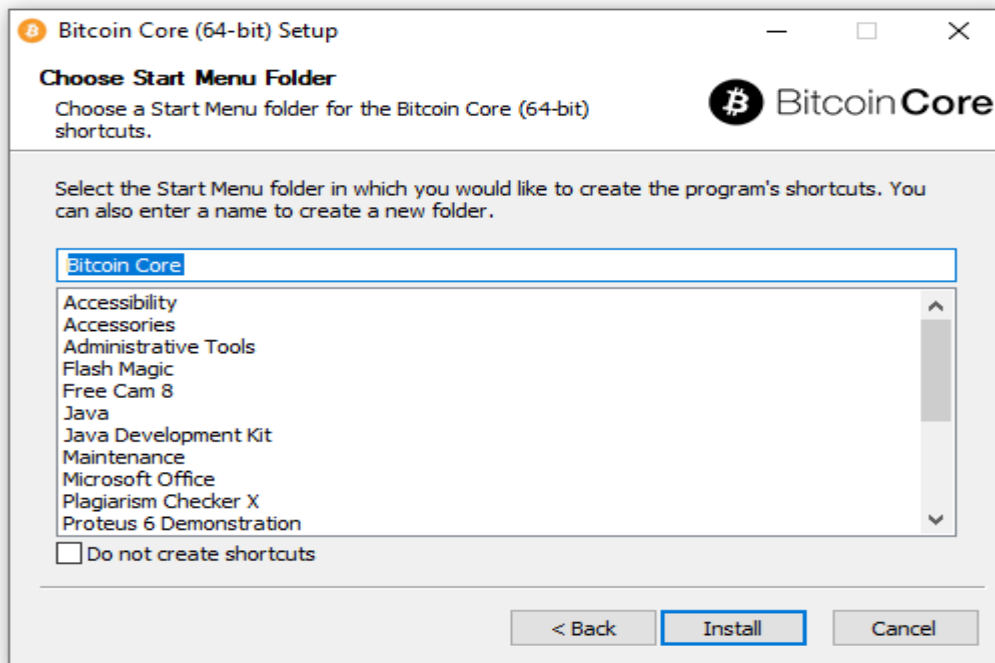
Step 3: Run the setup file-> click next

Step 4: Click Next

Step 5: Finally click on Install

Launch Bitcoin Core-> Click OK.

Welcome ✕

Welcome to Bitcoin Core.

As this is the first time the program is launched, you can choose where Bitcoin Core will store its data.

Bitcoin Core will download and store a copy of the Bitcoin block chain. Approximately 8 GB of data will be stored in this directory. The wallet will also be stored in this directory.

◉ Use the default data directory

◯ Use a custom data directory:

C:\Users\Mithilesh\AppData\Roaming\Bitcoin `...`

25 GB of free space available.

A new data directory will be created.

When you click OK, Bitcoin Core will begin to download and process the full Bitcoin block chain (420GB) starting with the earliest transactions in 2009 when Bitcoin initially launched.
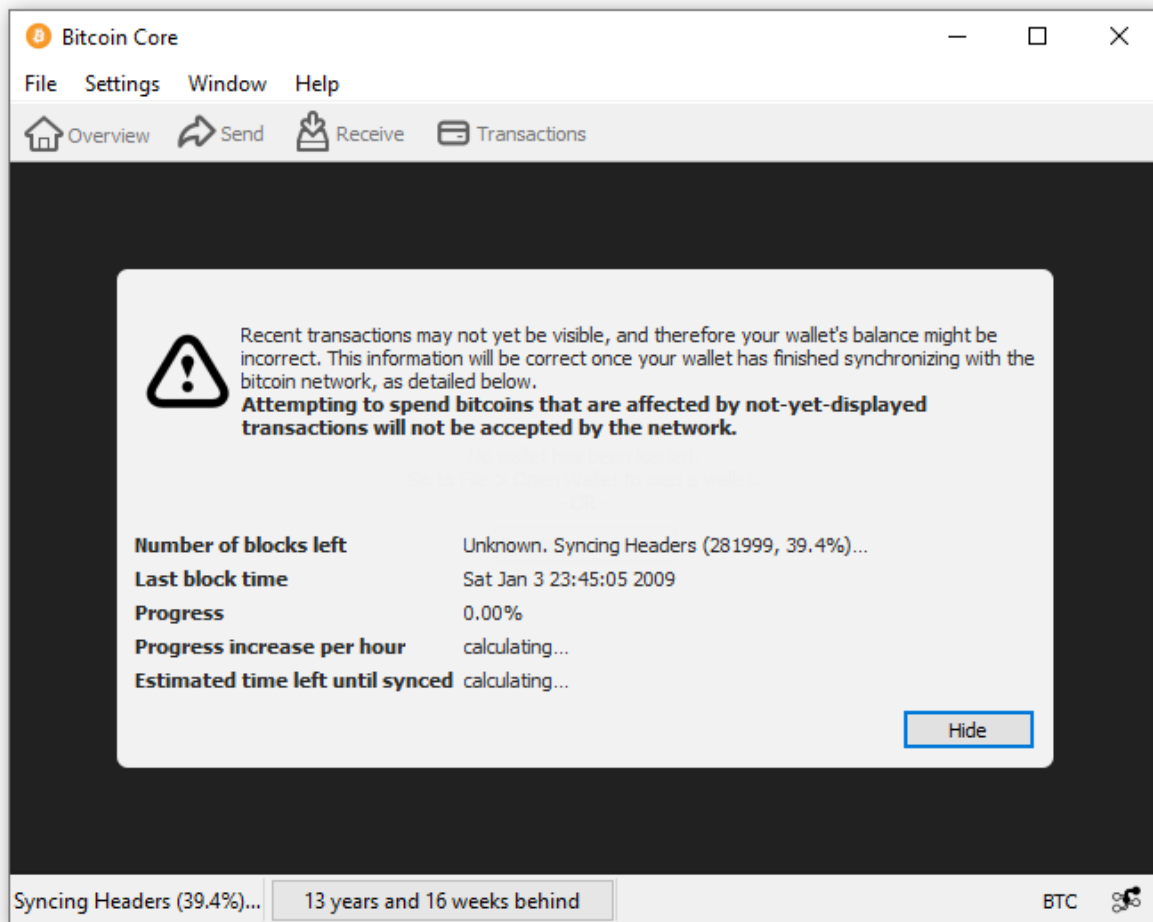
This initial synchronisation is very demanding, and may expose hardware problems with your computer that had previously gone unnoticed. Each time you run Bitcoin Core, it will continue downloading where it left off.

If you have chosen to limit block chain storage (pruning), the historical data must still be downloaded and processed, but will be deleted afterward to keep your disk usage low.
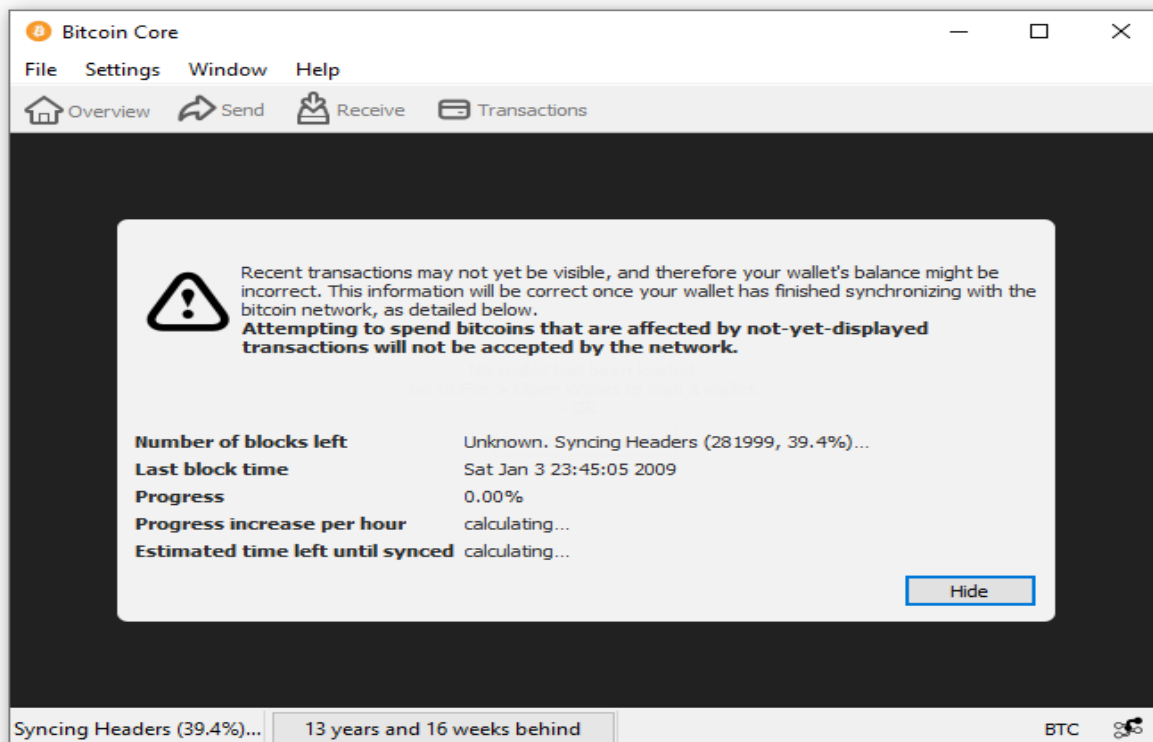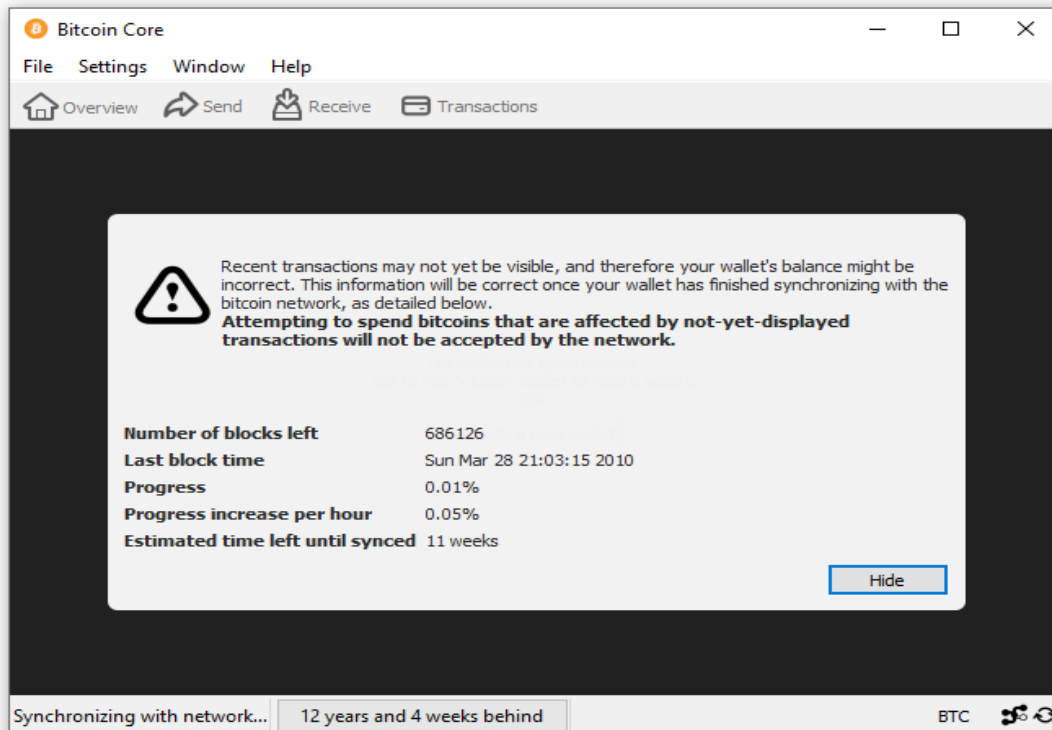
☑ Limit block chain storage to | 2 GB ⇅ | (sufficient to restore backups 6 days old)

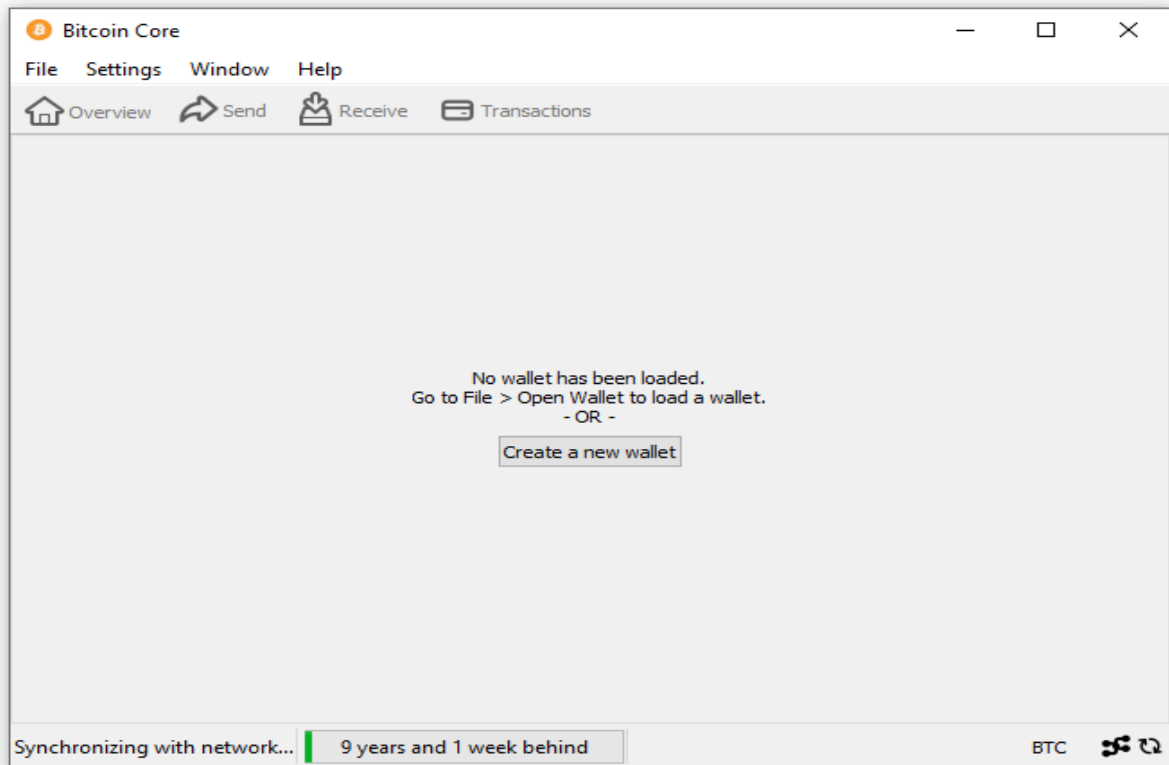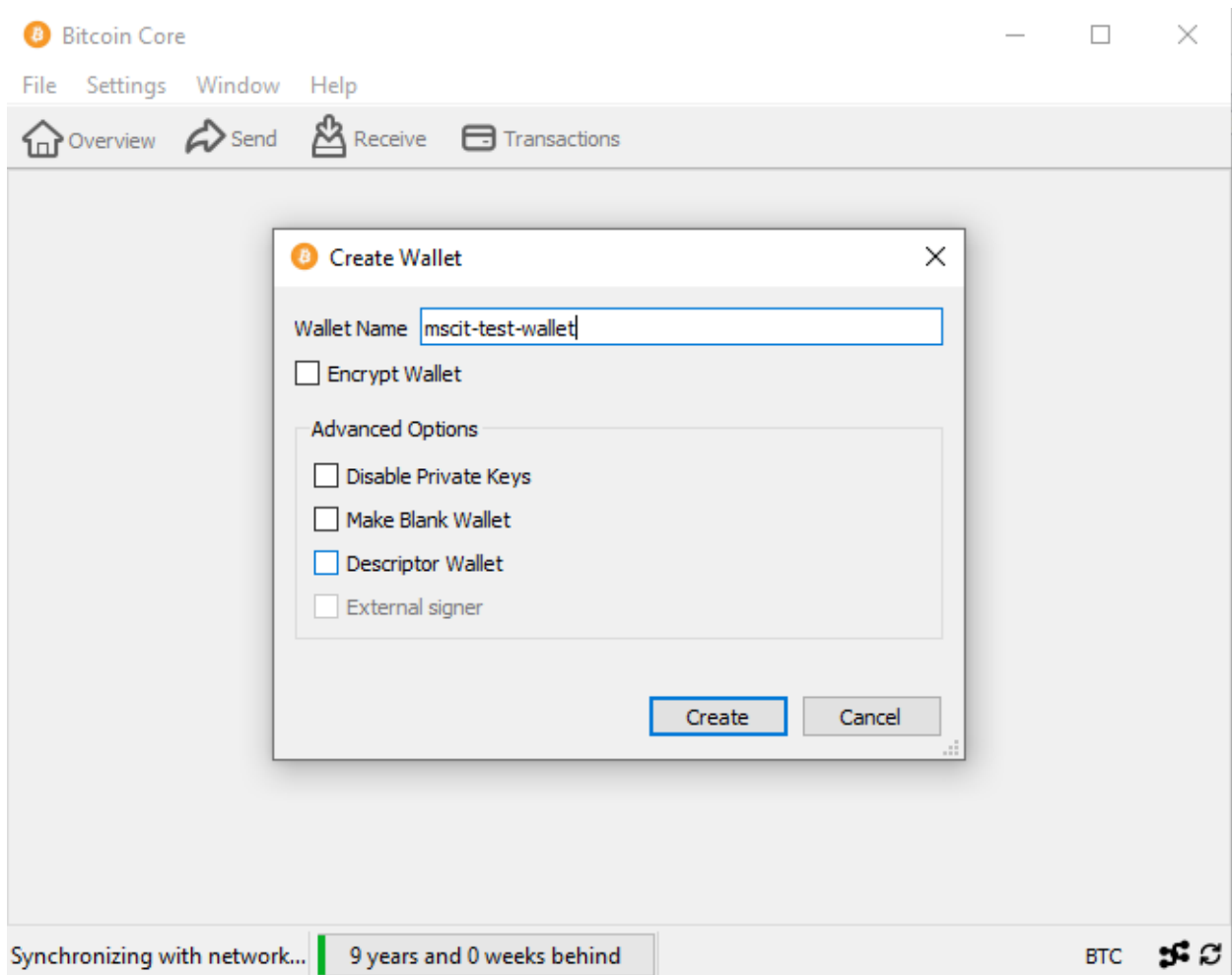OK   Cancel

Click on Hide button [Synchronization take place in background]

Bitcoin Core

File   Settings   Window   Help

🏠 Overview   ➡ Send   📩 Receive   💳 Transactions

⚠ Recent transactions may not yet be visible, and therefore your wallet's balance might be incorrect. This information will be correct once your wallet has finished synchronizing with the bitcoin network, as detailed below.
Attempting to spend bitcoins that are affected by not-yet-displayed transactions will not be accepted by the network.

Number of blocks left	686126
Last block time	Sun Mar 28 21:03:15 2010
Progress	0.01%
Progress increase per hour	0.05%
Estimated time left until synced	11 weeks

Hide

Synchronizing with network... | 12 years and 4 weeks behind | BTC

Bitcoin Core

File   Settings   Window   Help

🏠 Overview   ➡ Send   📩 Receive   💳 Transactions

⚠ Recent transactions may not yet be visible, and therefore your wallet's balance might be incorrect. This information will be correct once your wallet has finished synchronizing with the bitcoin network, as detailed below.
Attempting to spend bitcoins that are affected by not-yet-displayed transactions will not be accepted by the network.

Number of blocks left	Unknown. Syncing Headers (281999, 39.4%)...
Last block time	Sat Jan 3 23:45:05 2009
Progress	0.00%
Progress increase per hour	calculating...
Estimated time left until synced	calculating...

Hide

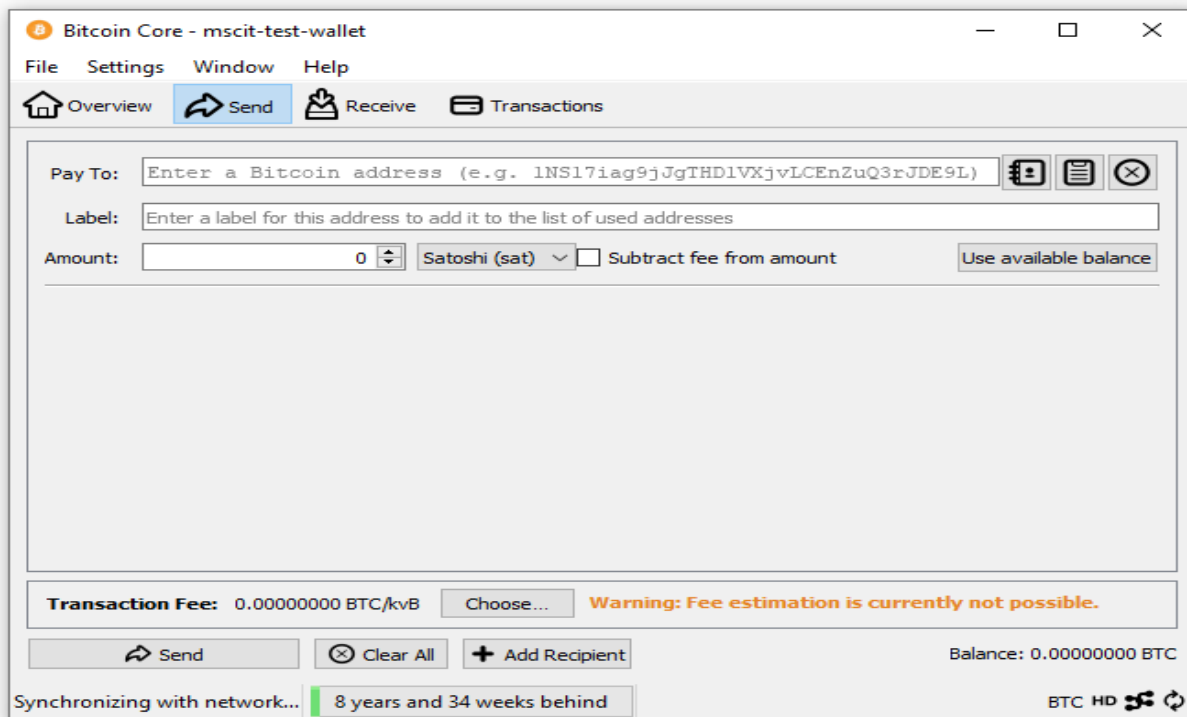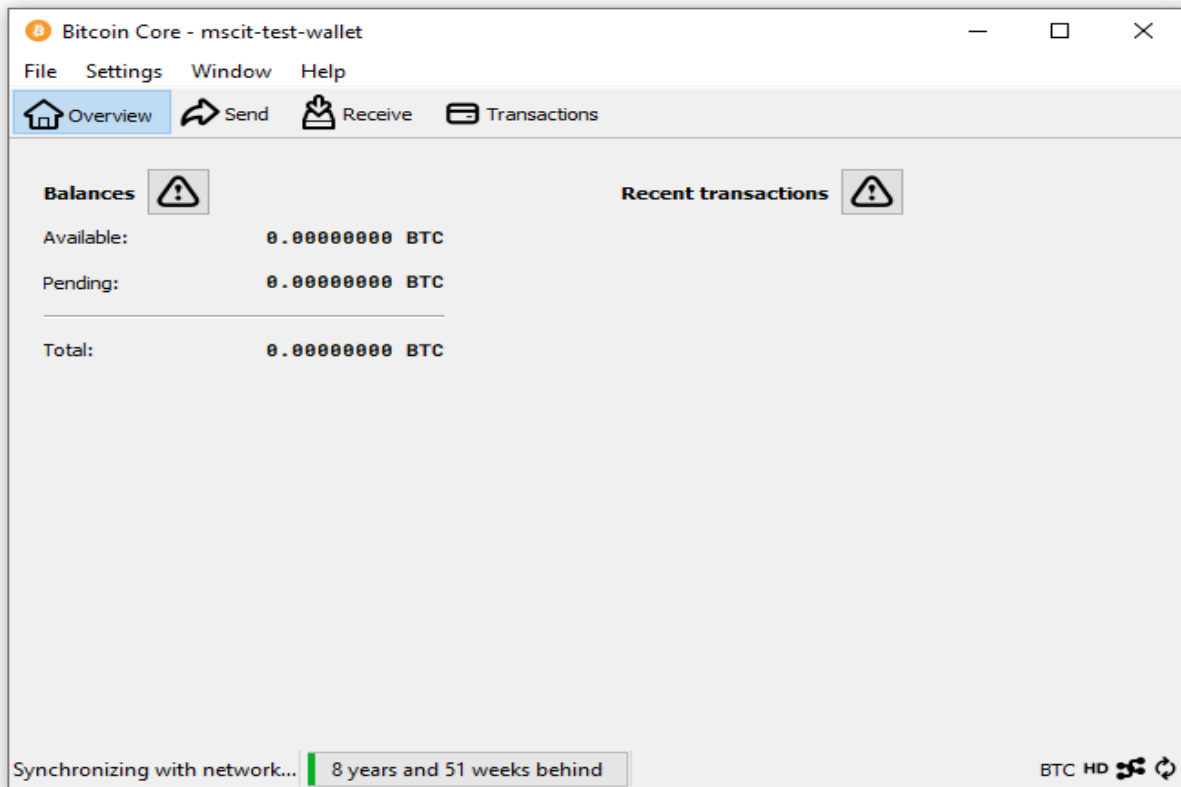Syncing Headers (39.4%)... | 13 years and 16 weeks behind | BTC

You can create a wallet -> Create a new wallet

Enter Wallet name

Finally Account is setup

Bitcoin Core - mscit-test-wallet ― ☐ ✕

File   Settings   Window   Help

🏠 Overview   ➤ Send   📥 Receive   ▭ Transactions

Balances ⚠

Available: 0.00000000 BTC

Pending: 0.00000000 BTC

────────────────────────────

Total: 0.00000000 BTC

Recent transactions ⚠

Synchronizing with network... | 8 years and 51 weeks behind | BTC **HD** ⚒ ↻

Bitcoin Core - mscit-test-wallet ― ☐ ✕

File   Settings   Window   Help

🏠 Overview   ➤ Send   📥 Receive   ▭ Transactions

Pay To: | Enter a Bitcoin address (e.g. 1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L) | 📇 📄 ⊗

Label: | Enter a label for this address to add it to the list of used addresses |

Amount: | 0 ⇕ | Satoshi (sat) ∨   ☐ Subtract fee from amount   Use available balance

Transaction Fee: 0.00000000 BTC/kvB | Choose... | Warning: Fee estimation is currently not possible.

| ➤ Send | | ⊗ Clear All | | ➕ Add Recipient | Balance: 0.00000000 BTC

Synchronizing with network... | 8 years and 34 weeks behind | BTC **HD** ⚒ ↻

Practical 8

Aim: Demonstrate the use of Bitcoin Core API.

Open Python IDLE and create new Script.

#######################

```
from bitcoinlib.wallets import Wallet
w = Wallet.create('Wallet6')
key1 = w.get_key()
print('Wallet Address:',key1.address)
w.scan()
print(w.info())
```

####################
Open CMD and install bitcoinlib package
pip install bitcoinlib

After installing package run the program from Python IDLE

```
IDLE Shell 3.9.5                                                              —    □    ×
File   Edit   Shell   Debug   Options   Window   Help
Wallet Addres  1BAtKdJbsim7mMTr5HKH5DtdeDvzssSojd
=== WALLET ===
 ID                              2
 Name                            Wallet6
 Owner
 Scheme                          bip32
 Multisig                        False
 Witness type                    legacy
 Main network                    bitcoin
 Latest update                   2022-04-27 19:29:29.811638

= Wallet Master Key =
 ID                              17
 Private                         True
 Depth                           0

- NETWORK: bitcoin -
- - Keys
    22 m/44'/0'/0'/0/0                     1BAtKdJbsim7mMTr5HKH5DtdeDvzssSojd
 address index 0                                   0.00000000 ฿
    23 m/44'/0'/0'/0/1                     15VAQ8xnRQJJscjq6qLFSXjEoMdQsB1Wxt
 address index 1                                   0.00000000 ฿
    24 m/44'/0'/0'/0/2                     1GvJ5gGdUGqbJR4iscxkkSVkuST9watsBm
 address index 2                                   0.00000000 ฿
    25 m/44'/0'/0'/0/3                     1KjMbCw82MPn2Ad4Gfg51JhcAfNRNYvepq
 address index 3                                   0.00000000 ฿
    26 m/44'/0'/0'/0/4                     1Md9b1xbajrRJQ1RvwDLE4RV3ZpAWLRFAZ
 address index 4                                   0.00000000 ฿
    28 m/44'/0'/0'/1/0                     1QLTktKig5rx6khsSX5kJBFkQ7rVnTG3Qd
 address index 0                                   0.00000000 ฿
    29 m/44'/0'/0'/1/1                     16fyVf7SSEobK1yVDj9NnCrbJPn2wnvF5A
 address index 1                                   0.00000000 ฿
    30 m/44'/0'/0'/1/2                     1Dz6zQ7PF57Hg2qU2fNHBEgwmrrPqKo16o
 address index 2                                   0.00000000 ฿
    31 m/44'/0'/0'/1/3                     1EMDowBLGdXVpMKn16Hnpeq1nMDNhZcQpn
 address index 3                                   0.00000000 ฿
    32 m/44'/0'/0'/1/4                     1EfQVH35druVHo5cRczPRbebNyVpv2uve9
```

More Detail: https://pypi.org/project/bitcoinlib/

Practical 9

Aim: Build Dapps with angular[using truffle and ganache cli]

Step 1:Install the required package –on new **terminal 1** type these commands

sudo apt-get -y install curl git vim build-essential

sudo apt-get install curl software-properties-common

sudo apt install npm

sudo npm install -g web3

sudo apt-get install nodejs

sudo apt install python3.9

curl -sL https://deb.nodesource.com/setup_10.x | sudo bash -

sudo npm install --global node-sass@latest

sudo npm install -g truffle@latest

sudo npm install -g ganache-cli

export NODE_OPTIONS=--openssl-legacy-provider

Step 2: Create a new directory

mkdir myproject

cd myproject

Step 3:Initialize the project folder

truffle init

#######################

////to update npm//

sudo npm cache clean -f

sudo npm install -g n

sudo n latest

#####################

Step 4: Now create a new contract

nano contracts/HelloWorld.sol

Step 5:Add the following code in HelloWorld.sol

pragma solidity ^0.8.0;

contract HelloWorld {

   function sayHello() public pure returns(string memory){

     return("hello world");

  }

}

Step 6:Edit default configuration file

nano migrations/1_initial_migration.js

Step 7: Edit this line in the file

const Migrations = artifacts.require("HelloWorld");

module.exports = function (deployer) {

 deployer.deploy(Migrations,"hello");

};

Step 8: Edit network configuration file

sudo nano truffle-config.js

Remove all line(press CTRL +K) from the file and add the following lines

########################

module.exports = {

  networks: {

   development: {

    host: "127.0.0.1",

    port: 8545,

    network_id: "*",

   }

  }

}

########################

Step 9: start ganache-cli –Switch/Open to **terminal 2**

ganache-cli

Step 10: deploy the truffle deploy- On **terminal 1**

truffle deploy

[Note contract address]

```
⊗ ⊖ ☐   mithilesh@ubuntu: ~/dapptest
> Network name:      'development'
> Network id:         1649516002204
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
=======================

   Deploying 'HelloWorld'
   ----------------------
   > transaction hash:      0x51a8203fb4972e9286
56b7ba531ea74f61231540714fa1983091b1077810e72d
   > Blocks: 0              Seconds: 0
   > contract address:      0xD9Fd118164b669E742
577A04378397A61837e2c9
   > block number:          1
   > block timestamp:       1649516161
   > account:               0x4Ff91151b98D04Ab14
751d47B910Dc2d7fB86f87
   > balance:               99.9972989
   > gas used:              135055 (0x20f8f)
   > gas price:             20 gwei
   > value sent:            0 ETH
   > total cost:            0.0027011 ETH
```

Step 11: Open truffle console - On **terminal 1**

truffle console

Step 11: Get reference of contact

**contract = await
HelloWorld.at('0x2C403EE1b30F56C0c773089c1Eb9DddF1499C969')**

[Replace '0x2C403EE1b30F56C0c773089c1Eb9DddF1499C969' with your contact
address; every time you compile/deploy a new contract address will be generated]

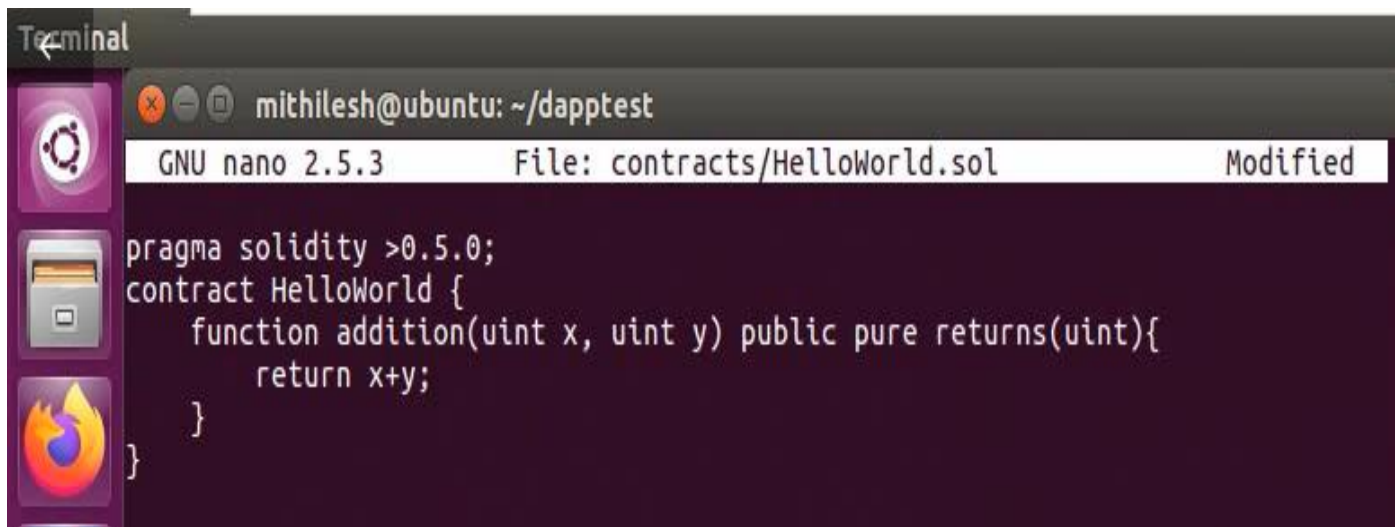Step 12: Call the function from the contract

a = await contract.sayHello()

Step 13:Print output on the screen

[In case you are getting any error for version; change the solidity version in HelloWorld.sol]

Example 2:

Save-> Deploy ->Open console