PRACTICAL NO 1

Write the following programs for blockchain in python:

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

CODE :

```
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
from Crypto import Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
class Client:
  def __init__(self):
    random = Crypto.Random.new().read
    self._private_key = RSA.generate(1024, random)
    self._public_key = self._private_key.publickey()
    self._signer = PKCS1_v1_5.new(self._private_key)
  @property
  def identity(self):
    idn = binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii') # <------
    return idn
Dinesh = Client()
print(Dinesh.identity)
```

OUTPUT :

```
========================== RESTART: F:/blockchain/la.py ==========================
30819f300d06092a864886f70d010101050003818d0030818902818100b1ba8aab2e7b5187148bd0
5713b2fbf7546943357a2b4b9156667e085760c4bb41cb715d03a7ff3277f62b7fde7591e42892af
760e065f291183fdc9e15d8f17092225292be41e2ca2216f51c7558f4cccc1ca339bb95f0a7803cc
475c276d5e74510ae53d64e425e8826059f07bd297dcb4fda7f6ebd2b40fd6085deae41211020301
0001
>>>
```

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

CODE :

```
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
from Crypto import Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer = PKCS1_v1_5.new(self._private_key)
    @property
    def identity(self):
        idn = binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii') # <------
        return idn
class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()
    def to_dict(self):
        if self.sender == "Genesis":
            identity = "Genesis"
        else:
```

```
            identity = self.sender.identity
        return collections.OrderedDict({'sender': identity,'recipient': self.recipient,'value': self.value,'time' : self.time})
    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')
Dinesh = Client()
Ramesh = Client()
t = Transaction(Dinesh, Ramesh.identity, 5.0)
signature = t.sign_transaction()
print (signature)
```

OUTPUT :

```
========================= RESTART: F:/blockchain/lb.py =========================
0b57205c2384feleecc4d752ce8fa6524876630afe34d9c41fe44da211e208fd384e64fb91551fa1
44a5f3dcad8157deeab471d02f0ccc2fa432f85f6d4a19e00bfe83094a7557f0142a91f503dc4c82
26b7dfc1429a82b1b2228956d7961bb238e8081801f488c497f7b80eb0f5336bb48a6ae3740ee08e
80707ce958678be0
>>>
```

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

CODE :

```
import hashlib
import json
from time import time

# creating the Block_chain class
class Block_chain(object):
    def __init__(self):
        self.chain = []
        self.pendingTransactions = []
        self.newBlock(previousHash = "The first Transaction", the_proof = 100)
    def newBlock(self, the_proof, previousHash = None):
        the_block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.pendingTransactions,
            'proof': the_proof,
            'previous_hash': previousHash or self.hash(self.chain[-1]),
```

```
        }
        self.pendingTransactions = []
        self.chain.append(the_block)

        return the_block

block_chain = Block_chain()
print("Genesis block: ", block_chain.chain)
```

OUTPUT :

```
========================== RESTART: F:/blockchain/ld.py ==========================
Genesis block:  [{'index': 1, 'timestamp': 1655141036.7629633, 'transactions': [
], 'proof': 100, 'previous_hash': 'The first Transaction'}]
>>>
```

E] Create a mining function and test it.

CODE :

```
# import libraries
import Crypto
from Crypto.PublicKey import RSA
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 hashlib

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

def mine(message, difficulty=1):
    assert difficulty >= 1
    prefix = '1' * difficulty
```

```
    for i in range(1000):
        digest = sha256(str(hash(message)) + str(i))
        if digest.startswith(prefix):
            print ("after " + str(i) + " iterations found nonce: "+ digest)
    return digest
mine ("test message", 2)
```

OUTPUT :

```
========================== RESTART: F:/blockchain/le.py ==========================
after 33 iterations found nonce: 11336a72b68d032585d1ad124492fc26e469e175bdee60c
fcf21ec783ed5ab72
after 704 iterations found nonce: 11dcfac3b2db19059cb2013e92826e94583aa138ecd0c2
029a78758a3d8ce519
after 964 iterations found nonce: 11fbf9a0fed8cbba028d3b0d0aa9cbe696c8809488af3b
abdd32aca7f3279acd
>>>
```