AMRITA VISHWA VIDYAPEETHAM

BLOCKCHAIN

19AIE431: APPLIED CRYPTOGRAPHY

TEAM – 14

PS GANESAN(CB.EN.U4CSE19233)

DEVISETTY MITHIL SRI SAI (CB.EN.U4AIE19023)

ROHITH G (CB.EN.U4AIE19026)

SHIVARUPAN S (CB.EN.U4AIE19059)

VIJAI SIMMON S (CB.EN.U4AIE19068)

Acknowledgements

Our team would like to extend our heartfelt gratitude to *Dr. K.P Soman*, who handled the course titled '*19AIE431 - Applied Cryptography*' and also was the guide to our project. His dedication and passion to teaching is one of the reasons that inspired us to take up a project in the previously unbeknownst field of Blockchains.

We would also like to give our hearty thanks to our university *Amrita Vishwa Vidyapeetham*, for providing us with this course thereby giving us a fantastic opportunity to explore and work in the field of cryptography.

We also want to show our love to our chancellor, *Devi Mata Amritanandamayi* with whose benevolence we could complete this project even in such trying times of the Covid-19 pandemic.

Table of Contents

Abstract

Blockchains are taking the world by storm. Starting from bank transactions to monitoring supply chains, blockchains are just everywhere. They provide lots of benefits: trustability, collaboration, organization, identification, credibility, and transparency.

Our project provides a basic implementation of a *blockchain-based voting system* in Python. The report begins with an introduction to the underlying concepts behind blockchains, like hashing, mining, and decentralization. It then proceeds to demonstrate the creation of a simple blockchain in Python. Python code snippets and detailed explanations have been given as and when required. The ensuing sections discuss the creation of a voting system on top of the previously built blockchain. The report then thoroughly analyses the pros and cons of using blockchain technology. It finally concludes by discussing how and why it is feasible to use blockchains in certain sectors.

Introduction to Blockchain

A *blockchain* refers to a group of blocks that contain information. The concept of blockchain is aimed at making blocks and their data tamper-resistant. It was originally used to store the timestamps of important digital documents so that predating or postdating those documents could be prevented. It was used by *Satoshi Nakamoto* to bring to existence the Bitcoin cryptocurrency.

A blockchain can also be considered as sort of a ledger in which anybody can make a one-time entry, i.e., it becomes *immutable* later. Changing the entry in the ledger would mean gaining the permission of all those who use the ledger.

Usually, every block in a blockchain has 3 fields: data, hash of the data, hash of the predecessor block. Depending on where they are used, the blocks can house different types of data: bank transaction history, customer credentials, medical records, tax records, and other important documents.

The security of a blockchain is derived from its *distributed nature*, i.e., there is no central entity managing the whole chain. In other words, all nodes/blocks are connected in a *peer-to-peer network*. Every node will have its own copy of the entire blockchain. Therefore, attempts to modify the data would mean modifying the blockchain stored in every node in the network which is extremely infeasible. Also, in order to add a block to the chain, every node in the chain has to give its *consensus* for the block to be added to its own copy of the chain. Another factor that adds to the security of a blockchain is the process of *mining*. It will be covered extensively in the sections that follow.

Hashing

In simple words, mapping input data to one amongst a very large set of numbers is called *hashing*. The number to which a given input is mapped is called the *hash* of the input.

Hashing is similar to *encryption* in the sense that they both convert an input to a completely different output. However, in encryption, the encrypted version of the data is usually the same size of the original data. **Ex:** in the character level encryption a text document.

This may not be true in the case of hashing. For instance, irrespective of the length of the input string, the SHA256 (which is also the hashing algorithm used in this project) returns only a 256-bit binary value.

Also, unlike in encryption wherein the original data can be retrieved, it would be impossible to convert the hash value obtained to original input data. In other words, hashing is a *one-way process*. This attribute of hashing is often used to ensure the *integrity* of messages in transit. Consider the following two use cases for instance:

1) **Document transfer:** Suppose a confidential text document is to be transferred. Firstly, the hash of the document is created. Next, along with the original document, an encrypted version of the hash (so that the hash itself cannot be modified in transit) is sent to the other side. The receiver first recomputes the hash of the document. The receiver then decrypts the encrypted hash and compares both values (the hash received and the hash recomputed). If the document had been modified in transit, then both hashes will obviously not match. In a similar fashion, hashes are used to verify *digital signatures*.

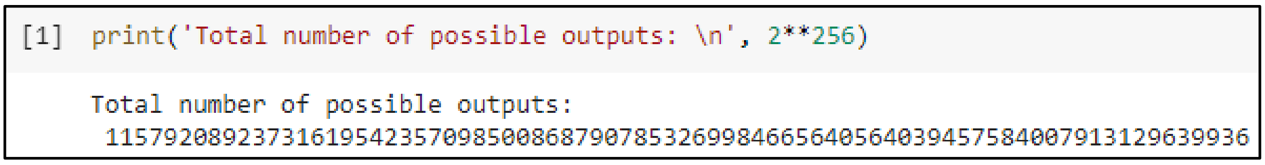
2) **Storing hashed passwords:** In order to verify whether the password entered by a user is correct, one might think that storing the passwords in a database and comparing it with the entered password is a solution. However, it is extremely risky since there is a chance of attackers to

discover the database. Therefore, only hashes of the actual passwords should be stored in the database and when a user enters their password, its hash will be computed and compared with the ones stored in the database.

As mentioned previously, our project uses *SHA-256* as the hashing algorithm.

The **SHA-256 (Secure Hash Algorithm-256)** belongs to the family of SHA-2 (Secure Hash Algorithm 2), which happens to be the successor of SHA-1. It is a patented cryptographic hash function that when given text inputs returns a 256 bit number.
In the cybersecurity realm, SHA-256 is used in implementing authentication and encryption protocols like SSL (Secure Socket Layer), TLS (Transport Layer Security), IPsec (Internet Protocol Security), SSh (Secure Shell), and PGP (Pretty Good Privacy). Unix uses SHA-256 for secure password hashing. Cryptocurrencies such as Bitcoin use SHA-256 for verifying transactions

The 'sha256' algorithm segments input data into chunks of 256 bits, performs a series of manipulations and maps the input to a 256 bit output (also to be noted is that for a given input, SHA256 always produces the same output). Therefore, the number of possible outputs is also 2^{256}.

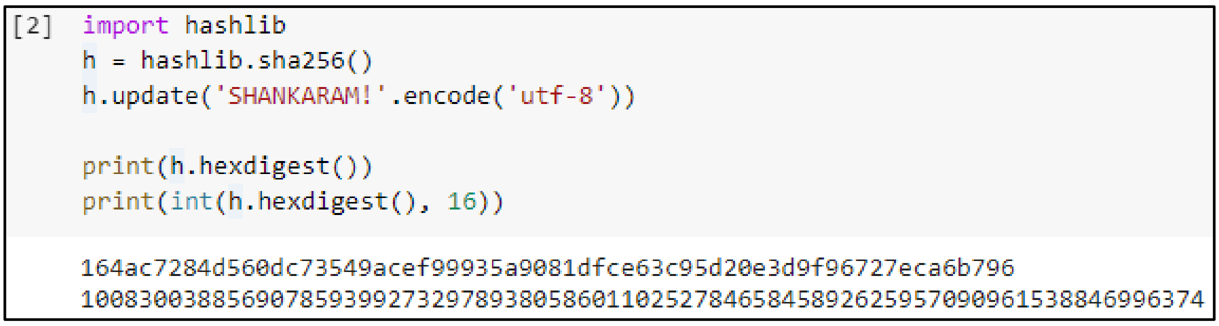


```
[1]  print('Total number of possible outputs: \n', 2**256)

     Total number of possible outputs:
      115792089237316195423570985008687907853269984665640564039457584007913129639936
```

The SHA256 hashing is done by importing the 'sha256' class from Python's *hashlib* library and creating an object of it. In the code below, 'h' is a SHA256 object. The string 'SHANKARAM' encoded in UTF-8 format is the data passed.

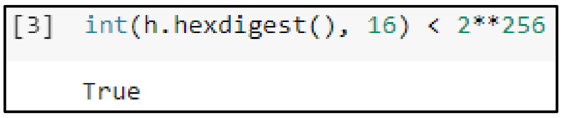
UTF-8 (Unicode Transformation Format – 8-bit), which is the most popular way to represent Unicode text in web pages and databases, is a variable-width character encoding used for electronic communication. It follows the Unicode Standard. It maps a given Unicode character to a unique binary string, and when given the binary string, it can map it back to the corresponding Unicode character.

Calling the *hexdigest()* function on the object ‘h’ returns the hash value in hexadecimal. Its decimal equivalent can be obtained using the *int()* function and passing the base as 16:



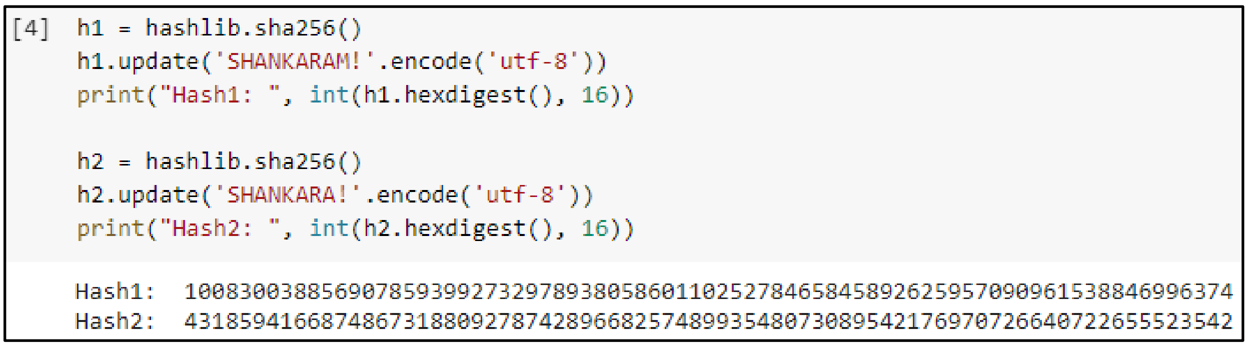
```
[2]  import hashlib
     h = hashlib.sha256()
     h.update('SHANKARAM!'.encode('utf-8'))

     print(h.hexdigest())
     print(int(h.hexdigest(), 16))

     164ac7284d560dc73549acef99935a9081dfce63c95d20e3d9f96727eca6b796
     10083003885690785939927329789380586011025278465845892625957090961538846996374
```

As can be seen, the integer equivalent is less than the highest possible value:



```
[3]  int(h.hexdigest(), 16) < 2**256

     True
```

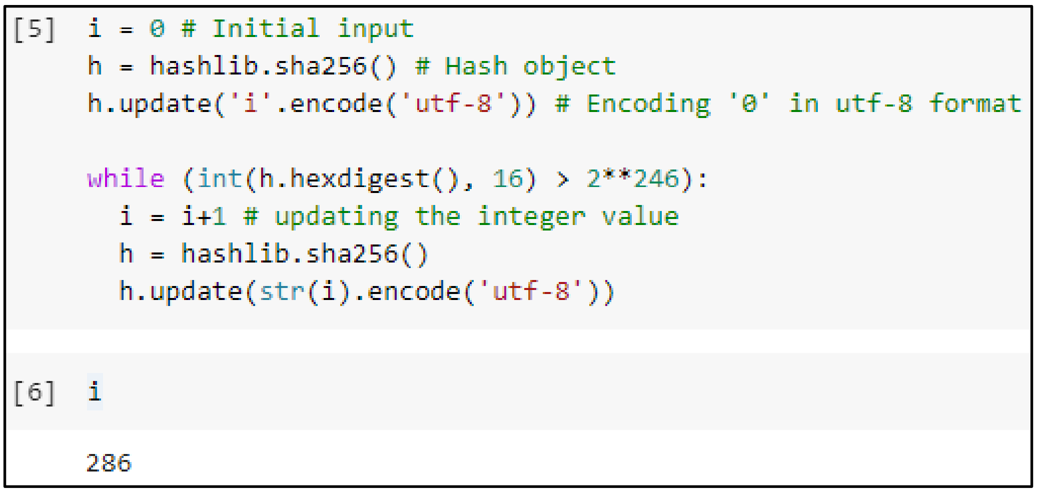
Even if two inputs are very similar, the corresponding hashes can be very different. For instance, the below two inputs very much resemble one another, however, their hashes are very different:



```
[4]  h1 = hashlib.sha256()
     h1.update('SHANKARAM!'.encode('utf-8'))
     print("Hash1: ", int(h1.hexdigest(), 16))

     h2 = hashlib.sha256()
     h2.update('SHANKARA!'.encode('utf-8'))
     print("Hash2: ", int(h2.hexdigest(), 16))

     Hash1:  10083003885690785939927329789380586011025278465845892625957090961538846996374
     Hash2:  43185941668748673188092787428966825748993548073089542176970726640722655523542
```

The vice versa is also possible. Therefore, given two hash values, one cannot guess whether the corresponding inputs will be similar to each other or not.

Mining a hash: Instead of directly generating a hash value and storing it in a block, some level of *difficulty* can be added to the process of block creation. For instance, there can be a condition that only hash values whose integer equivalent is less than $2^{(256-10)}$ shall be accepted. To understand the concept of difficulty, look at the example below:



```
[5]  i = 0 # Initial input
     h = hashlib.sha256() # Hash object
     h.update('i'.encode('utf-8')) # Encoding '0' in utf-8 format

     while (int(h.hexdigest(), 16) > 2**246):
       i = i+1 # updating the integer value
       h = hashlib.sha256()
       h.update(str(i).encode('utf-8'))
```

```
[6]  i

     286
```

Here, integers starting from 0 are first encoded as strings, i.e, “0”, “1”, “2”, “3”, etc. These are passed as inputs to the hash function. The *while* loop will terminate only when the integer equivalent of the hash lies below $2^{(256 - 10)}$. It is observed to stop when ‘i’ equals 286. In other words, the integers from 0 to 285 will not have their corresponding integer hash less than $2^{(256 - 10)}$. Thus, i = 286 is the value that clears the difficulty level.

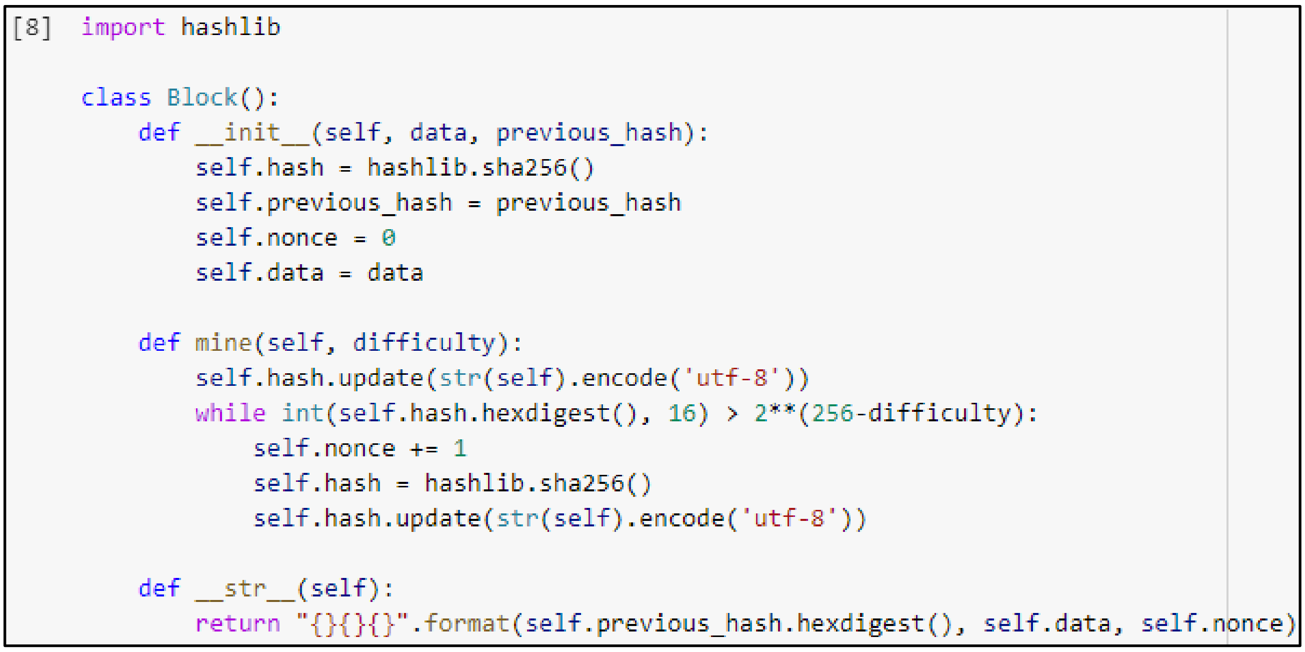
Note: In the above *while* loop, a new hash object is created in every iteration. If the same hash object were getting updated with the integer ‘i’, then the data in the k^{th} iteration would represent the cumulative collection of all the data passed in till the k^{th} iteration.

This process of adding constraints to the process of generating hashes is called ***mining***. As will be seen in the further sections, mining is done during block creation to add security to the blockchain. Also, the more difficult it is to mine a block, the more valuable it becomes.

Creating a Blockchain in Python

class ‘Block’

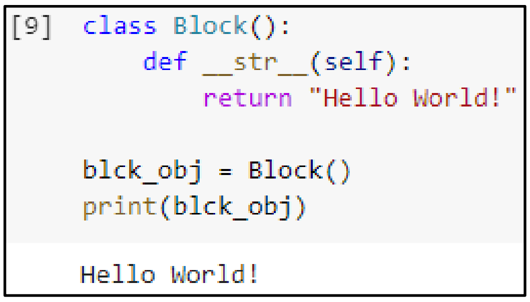
Creating a class for a block in a blockchain is similar to creating a class for a node in a linked list. Shown below is the class definition for a Block:



```
[8]  import hashlib

     class Block():
         def __init__(self, data, previous_hash):
             self.hash = hashlib.sha256()
             self.previous_hash = previous_hash
             self.nonce = 0
             self.data = data

         def mine(self, difficulty):
             self.hash.update(str(self).encode('utf-8'))
             while int(self.hash.hexdigest(), 16) > 2**(256-difficulty):
                 self.nonce += 1
                 self.hash = hashlib.sha256()
                 self.hash.update(str(self).encode('utf-8'))

         def __str__(self):
             return "{}{}{}".format(self.previous_hash.hexdigest(), self.data, self.nonce)
```

1) **__init__ method:** Each block will have the following attributes: *data, hash of that data*, *hash of the previous block*, and a *nonce* (short for number only used once) value. When a block is created, its nonce will be set to zero.

2) **__str__ method:** Similar to how the __init__ method is called by default when a block object is created, the __str__ is called when the block instance is passed to a function that accepts string arguments. For instance, consider the print() function. It accepts strings as arguments. So, when print(block_instance) is executed, the code inside the __str__ function of that block instance will get executed. An example with a simple class definition is given below:

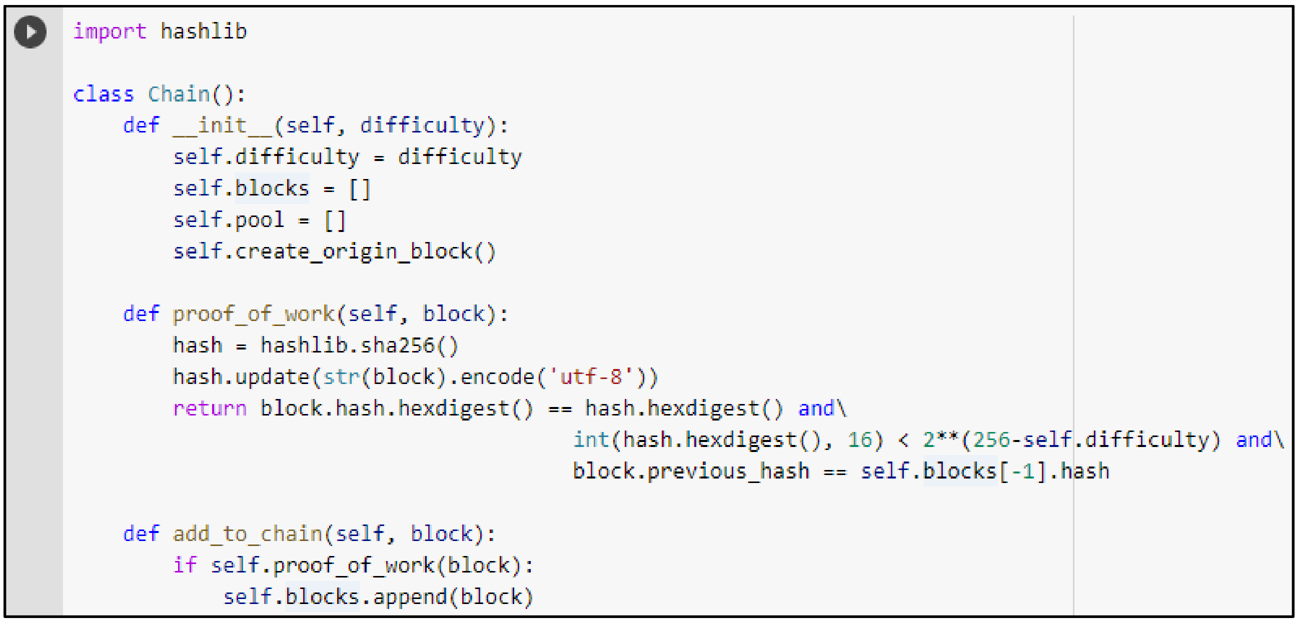


```
[9]  class Block():
         def __str__(self):
             return "Hello World!"

     blck_obj = Block()
     print(blck_obj)

     Hello World!
```

3) **mine() method**: Calling the mine() function on the block object will update the nonce to a positive integer that will satisfy the difficulty condition.

class ‘Chain’

In a *linked list*, apart from the class that creates *node* objects, there is another class that takes in the node objects and returns a traversable list object. Similarly, in blockchain too, a class is defined to implement the chaining of block objects. Let it be called *Chain*. Assume ‘*difficulty*’ is the same for all blocks in the chain. Therefore it can be treated as a common parameter for the entire blockchain and therefore passed as an argument to the __init__ method of *Chain*.



```python
import hashlib

class Chain():
    def __init__(self, difficulty):
        self.difficulty = difficulty
        self.blocks = []
        self.pool = []
        self.create_origin_block()

    def proof_of_work(self, block):
        hash = hashlib.sha256()
        hash.update(str(block).encode('utf-8'))
        return block.hash.hexdigest() == hash.hexdigest() and\
                                        int(hash.hexdigest(), 16) < 2**(256-self.difficulty) and\
                                        block.previous_hash == self.blocks[-1].hash

    def add_to_chain(self, block):
        if self.proof_of_work(block):
            self.blocks.append(block)
```

1) **__init__ method:** The ‘*blocks*’ attribute is initialized as an empty list. It will be updated as and when blocks are mined and added. ‘*pool*’ stores the data that is to be added to the blockchain. Once the corresponding block that will hold the data is mined and ready, the data will be erased from ‘*pool*’ and stored in the block.

2) **proof_of_work method:** Before getting added to the chain, an incoming block needs to be tested for a few conditions:

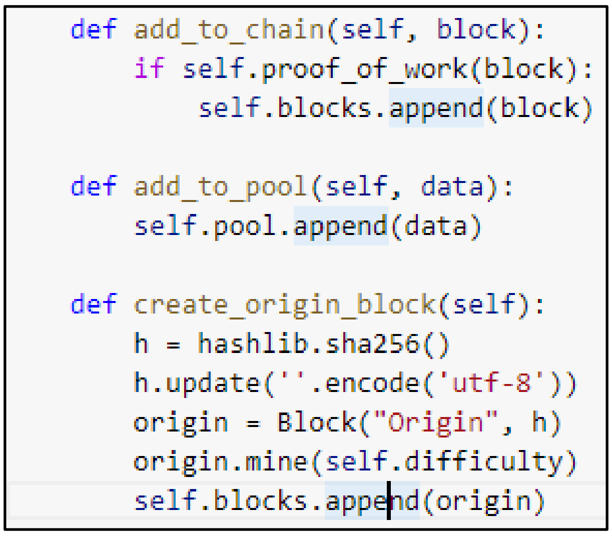
   a) Whether the *hash* that the block holds is actually the hash of the data
   b) Whether the *difficulty* condition has been satisfied

c) Whether the *'previous_hash'* of the block actually points to the hash of the block before it.

The method 'proof_of_work' returns true only if all these 3 conditions are satisfied.

<u>Note:</u> In fact, the 3rd condition is what implements ***security*** in the blockchain. If attackers try to alter the data in the first block, then its hash will not match the *'previous_hash'* of the second block. In other words, the second block would no longer be 'linked' to the first one. Attackers cannot cover this up simply by changing the *'previous_hash'* of the second block because then the hash of the second block would not match with the *'previous_hash'* of the 3rd block and so on. Therefore, changing data in one block would mean having to alter all the blocks that follow or at least the majority of the blocks.

One might think that It is easy to to recompute the hashes and *'previous_hash'es* of the blocks that follow. However, the concept of *mining* and setting a high value of *difficulty* can make it extremely time-consuming to modify and add even a single block, not to mention hundreds and thousands of blocks.

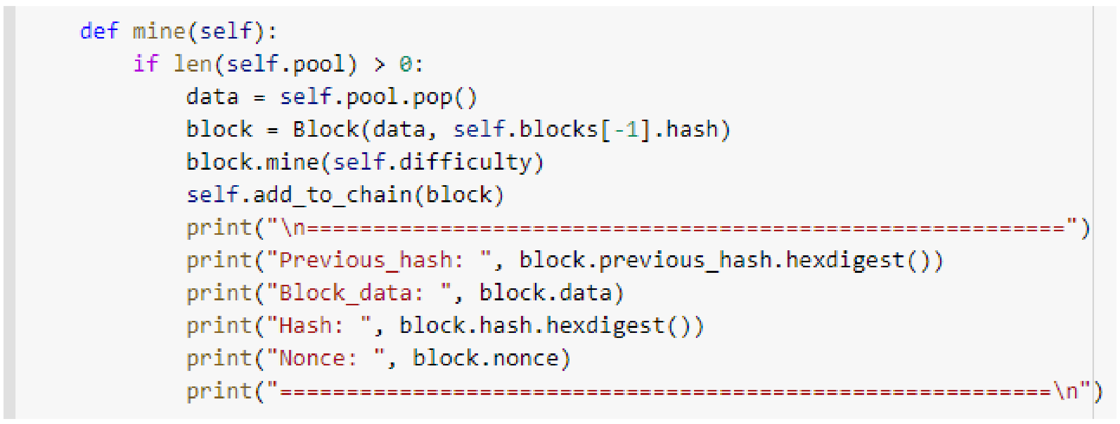


```
    def add_to_chain(self, block):
        if self.proof_of_work(block):
            self.blocks.append(block)

    def add_to_pool(self, data):
        self.pool.append(data)

    def create_origin_block(self):
        h = hashlib.sha256()
        h.update(''.encode('utf-8'))
        origin = Block("Origin", h)
        origin.mine(self.difficulty)
        self.blocks.append(origin)
```

3) **add_to_chain method:** If proof_of_work returns true, then this method adds the incoming block to the chain instance.

4) **add_to_pool method:** Adds the given data to the 'pool' attribute of the chain instance.

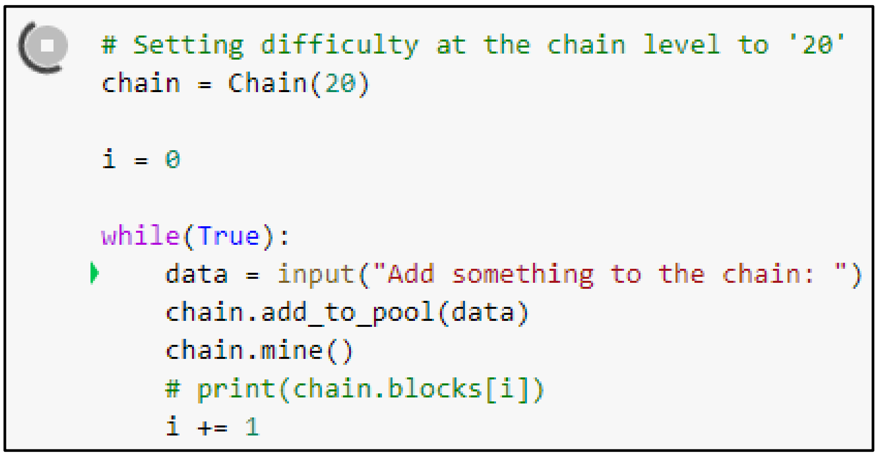
5) **create_origin_block method:** The class attributes demand that creation of a block needs a previous hash. The very first block in the chain is no exception to this. However, while passing a block to an empty chain, there can be no previous block and hence no previous hash. Therefore, even before the user passes blocks to be added to the chain, a ***genesis block*** or ***block zero*** needs to be present. This is implemented using the *create_origin_block* method. Note that the *proof_of_work* method is not called while creating the genesis block since calling it checks for the *previous_hash* condition (the genesis block does not have a predecessor). Consequently, *add_to_chain* cannot be called too. This requires that the genesis block be inserted into the chain manually.



```
    def mine(self):
        if len(self.pool) > 0:
            data = self.pool.pop()
            block = Block(data, self.blocks[-1].hash)
            block.mine(self.difficulty)
            self.add_to_chain(block)
            print("\n============================================================")
            print("Previous_hash: ", block.previous_hash.hexdigest())
            print("Block_data: ", block.data)
            print("Hash: ", block.hash.hexdigest())
            print("Nonce: ", block.nonce)
            print("============================================================\n")
```

6) **mine() method:** If there is any data in the *'pool'* list, *pop()* is called and the last element (corresponding to the index '-1') is taken to create a block instance. The *mine()* function of the block instance is called and the new block will be added to the chain and its contents neatly printed.

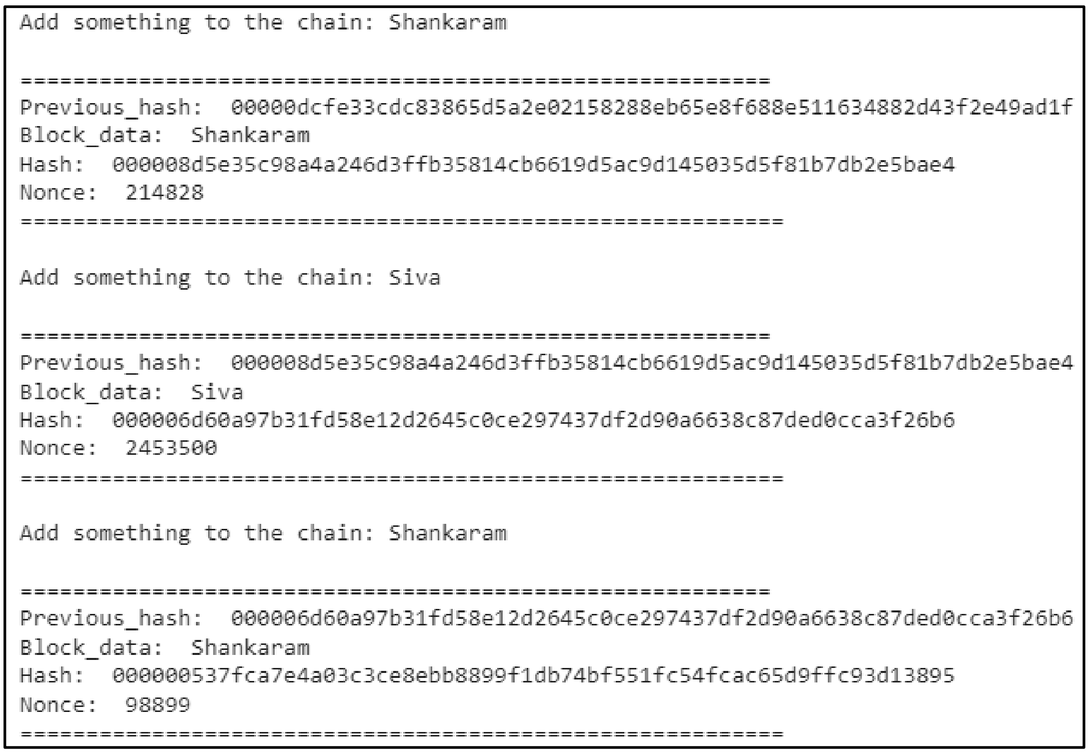
Main codescript



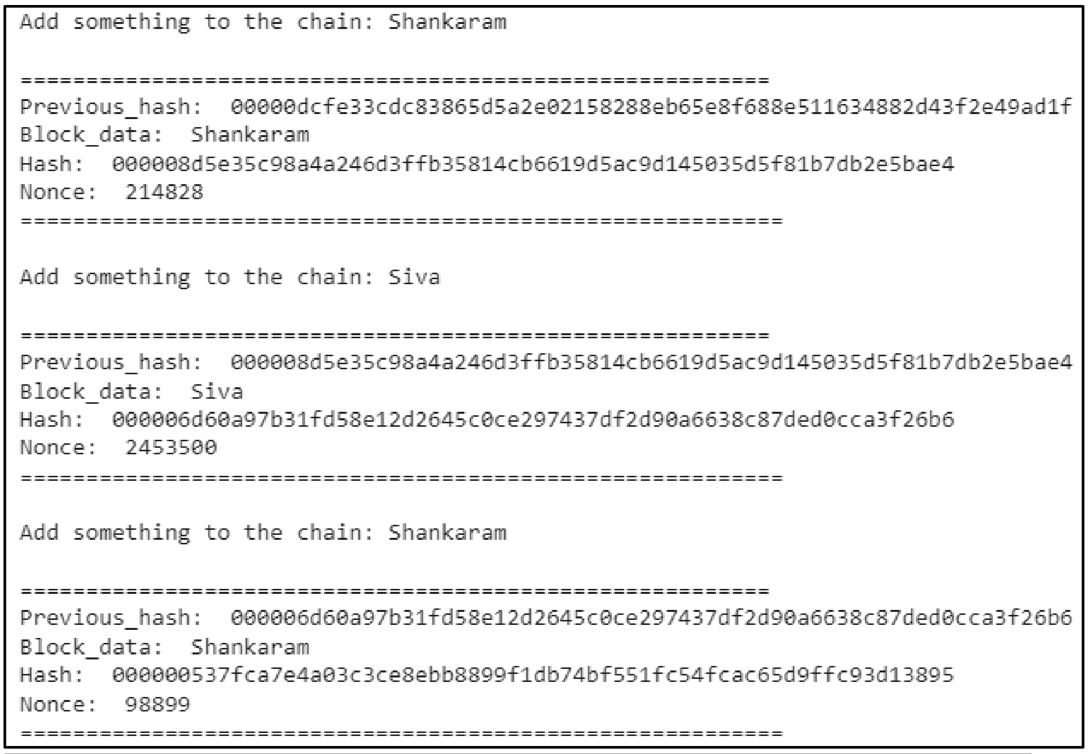
```python
# Setting difficulty at the chain level to '20'
chain = Chain(20)

i = 0

while(True):
    data = input("Add something to the chain: ")
    chain.add_to_pool(data)
    chain.mine()
    # print(chain.blocks[i])
    i += 1
```

The following code script uses the objects of the class ‘Block’ and ‘Chain’ to create a blockchain. It does so using an infinite while loop. In every iteration, the user is asked to add some information to the chain. This information is added to the *‘pool’* and once a new block is mined, it is transferred from the *‘pool’* to the block.

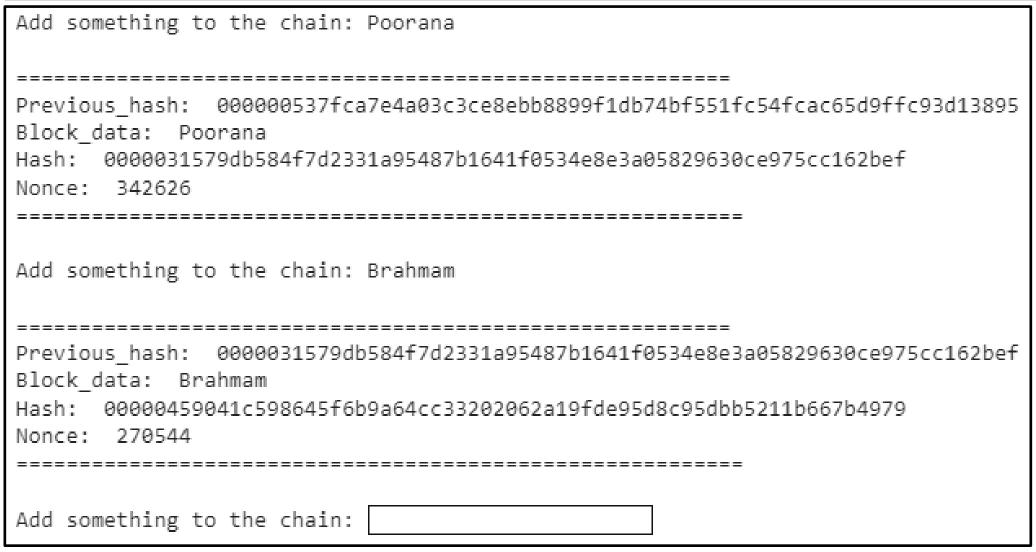
Output: Functioning of the blockchain



```
Add something to the chain: Shankaram

=========================================================
Previous_hash:  00000dcfe33cdc83865d5a2e02158288eb65e8f688e511634882d43f2e49ad1f
Block_data:  Shankaram
Hash:  000008d5e35c98a4a246d3ffb35814cb6619d5ac9d145035d5f81b7db2e5bae4
Nonce:  214828
==========================================================

Add something to the chain: Siva

=========================================================
Previous_hash:  000008d5e35c98a4a246d3ffb35814cb6619d5ac9d145035d5f81b7db2e5bae4
Block_data:  Siva
Hash:  000006d60a97b31fd58e12d2645c0ce297437df2d90a6638c87ded0cca3f26b6
Nonce:  2453500
==========================================================

Add something to the chain: Shankaram

=========================================================
Previous_hash:  000006d60a97b31fd58e12d2645c0ce297437df2d90a6638c87ded0cca3f26b6
Block_data:  Shankaram
Hash:  000000537fca7e4a03c3ce8ebb8899f1db74bf551fc54fcac65d9ffc93d13895
Nonce:  98899
==========================================================
```



```
Add something to the chain: Shankaram

========================================================
Previous_hash:  00000dcfe33cdc83865d5a2e02158288eb65e8f688e511634882d43f2e49ad1f
Block_data:  Shankaram
Hash:  000008d5e35c98a4a246d3ffb35814cb6619d5ac9d145035d5f81b7db2e5bae4
Nonce:  214828
=========================================================

Add something to the chain: Siva

========================================================
Previous_hash:  000008d5e35c98a4a246d3ffb35814cb6619d5ac9d145035d5f81b7db2e5bae4
Block_data:  Siva
Hash:  000006d60a97b31fd58e12d2645c0ce297437df2d90a6638c87ded0cca3f26b6
Nonce:  2453500
=========================================================

Add something to the chain: Shankaram

========================================================
Previous_hash:  000006d60a97b31fd58e12d2645c0ce297437df2d90a6638c87ded0cca3f26b6
Block_data:  Shankaram
Hash:  000000537fca7e4a03c3ce8ebb8899f1db74bf551fc54fcac65d9ffc93d13895
Nonce:  98899
=========================================================
```

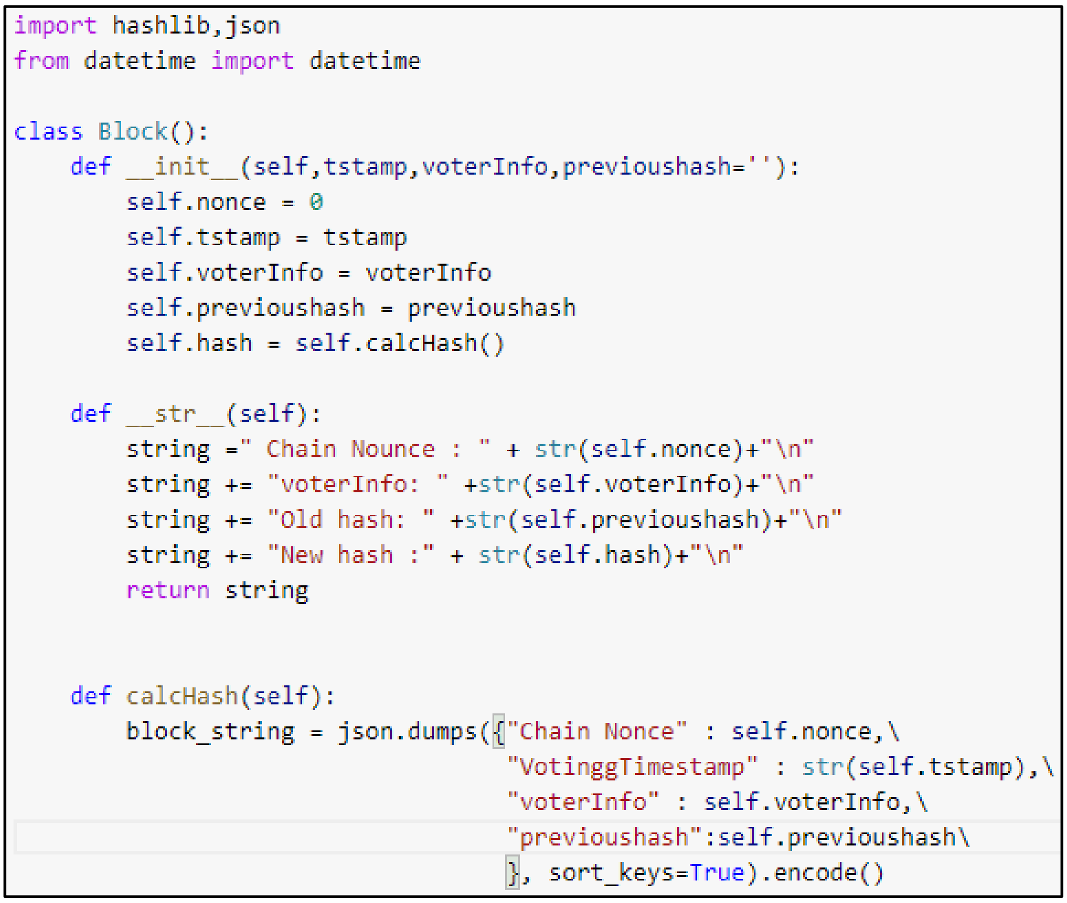


```
Add something to the chain: Poorana

========================================================
Previous_hash:  000000537fca7e4a03c3ce8ebb8899f1db74bf551fc54fcac65d9ffc93d13895
Block_data:  Poorana
Hash:  0000031579db584f7d2331a95487b1641f0534e8e3a05829630ce975cc162bef
Nonce:  342626
=========================================================

Add something to the chain: Brahmam

========================================================
Previous_hash:  0000031579db584f7d2331a95487b1641f0534e8e3a05829630ce975cc162bef
Block_data:  Brahmam
Hash:  00000459041c598645f6b9a64cc33202062a19fde95d8c95dbb5211b667b4979
Nonce:  270544
=========================================================

Add something to the chain: 
```

A simple blockchain-based voting system

The following sections detail the implementation of a simple voting system using blockchain.

class 'Block'



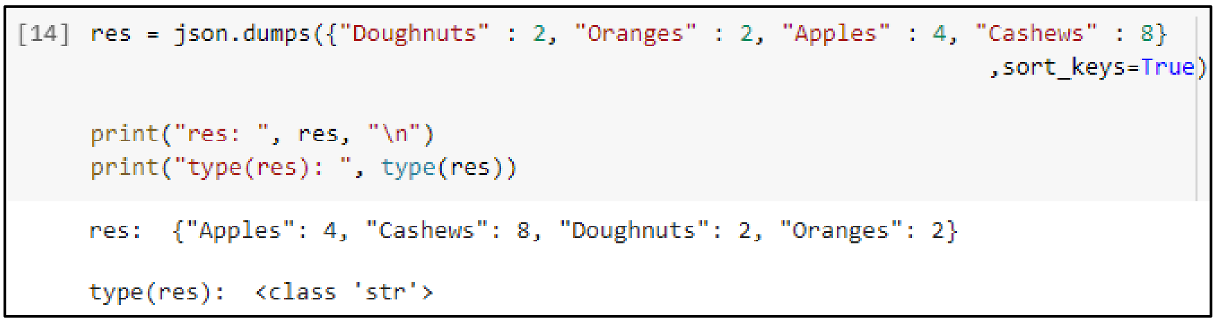
```python
import hashlib,json
from datetime import datetime

class Block():
    def __init__(self,tstamp,voterInfo,previoushash=''):
        self.nonce = 0
        self.tstamp = tstamp
        self.voterInfo = voterInfo
        self.previoushash = previoushash
        self.hash = self.calcHash()

    def __str__(self):
        string =" Chain Nounce : " + str(self.nonce)+"\n"
        string += "voterInfo: " +str(self.voterInfo)+"\n"
        string += "Old hash: " +str(self.previoushash)+"\n"
        string += "New hash :" + str(self.hash)+"\n"
        return string


    def calcHash(self):
        block_string = json.dumps({"Chain Nonce" : self.nonce,\
                                   "VotinggTimestamp" : str(self.tstamp),\
                                   "voterInfo" : self.voterInfo,\
                                   "previoushash":self.previoushash\
                                   }, sort_keys=True).encode()
```

1) **__init__ method:** Apart from the usual parameters namely nonce, previous hash, and hash value of the present block, every block in this case stores two additional parameters: voter's choice, and timestamp of vote casting.

2) **__str__ method:** This method concatenates the attributes in the order: nonce, voter information, current hash, and previous hash. It finally returns the concatenated string.

3) **calcHash() method:** The json.dumps() method converts a Python object into a *json* string. Setting the *sort_keys* parameter to *True* will

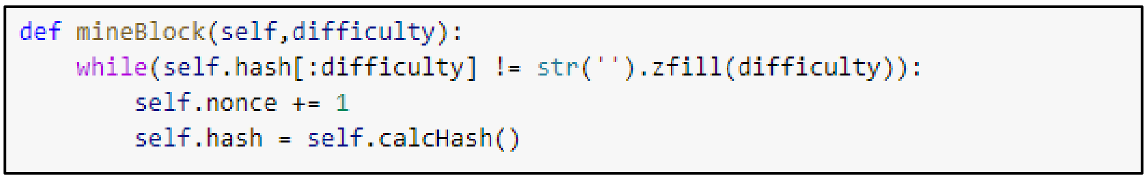
sort the dictionary based on keys. The below example illustrates the working of *json.dumps()* method:



```
[14] res = json.dumps({"Doughnuts" : 2, "Oranges" : 2, "Apples" : 4, "Cashews" : 8}
                                                                    ,sort_keys=True)


     print("res: ", res, "\n")
     print("type(res): ", type(res))

     res:  {"Apples": 4, "Cashews": 8, "Doughnuts": 2, "Oranges": 2}

     type(res):  <class 'str'>
```

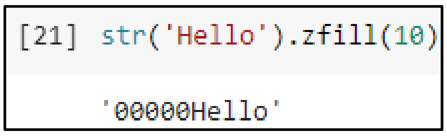
Note that the keys have been sorted in the alphabetical order after setting *sort_keys = True*

In the method definition, the *encode()* method has been called on the result obtained from *json.dumps()*. Therefore, the resultant string will be encoded in *UTF-8* format (*UTF-8* is the default parameter when no encoding scheme is specified to *encode()*)



```
def mineBlock(self,difficulty):
    while(self.hash[:difficulty] != str('').zfill(difficulty)):
        self.nonce += 1
        self.hash = self.calcHash()
```

4) **mineBlock() method:** Knowing the *zfill()* method will help understand how *mineBlock()* is defined. The *zfill()* method prepends the necessary amount of zeros to a string in order to match the desired output length. For example:



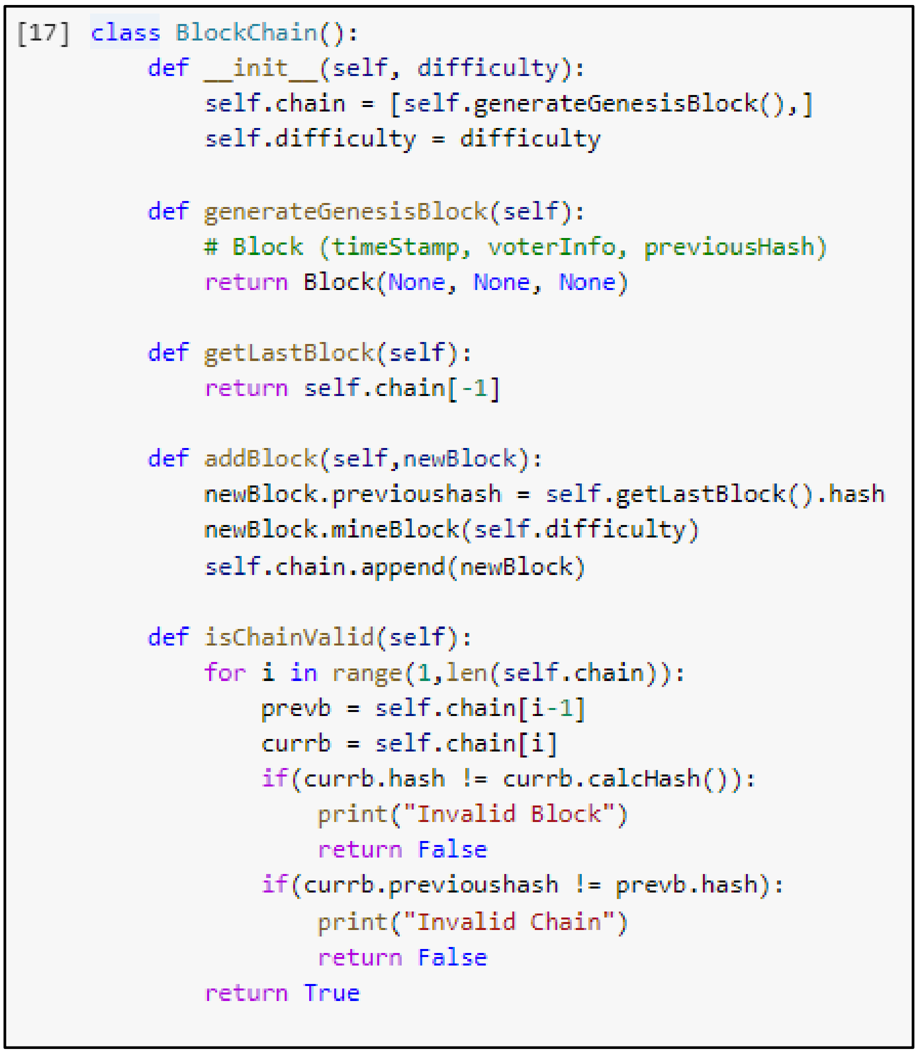
```
[21] str('Hello').zfill(10)

      '00000Hello'
```

If '*difficulty*' is 'n', then the statement *str('').zfill(difficulty)* will return a string with 'n' zeros.

Therefore, if the '*difficulty*' value is 'n', then it means that the 'n' leftmost values in the *hexdigest* of a block should be all zeros. In other words, the *while* loop will iterate as long as the leftmost 'n' values in the hexdigest is not a string of ‘n’ zeros. Once the leftmost ‘n’ values in the hexdigest become 0s, then the loop terminates and the corresponding hash and nonce values will be stored.

class ‘Blockchain’

This class takes in Block objects and creates a traversable list of blocks.



```
[17] class BlockChain():
         def __init__(self, difficulty):
             self.chain = [self.generateGenesisBlock(),]
             self.difficulty = difficulty

         def generateGenesisBlock(self):
             # Block (timeStamp, voterInfo, previousHash)
             return Block(None, None, None)

         def getLastBlock(self):
             return self.chain[-1]

         def addBlock(self,newBlock):
             newBlock.previoushash = self.getLastBlock().hash
             newBlock.mineBlock(self.difficulty)
             self.chain.append(newBlock)

         def isChainValid(self):
             for i in range(1,len(self.chain)):
                 prevb = self.chain[i-1]
                 currb = self.chain[i]
                 if(currb.hash != currb.calcHash()):
                     print("Invalid Block")
                     return False
                 if(currb.previoushash != prevb.hash):
                     print("Invalid Chain")
                     return False
             return True
```

1) **__init__ method:** When a blockchain instance is created, it needs to have a genesis block as mentioned before.

2) **generateGenesisBlock() method:** The genesis block will not take in any inputs for timestamp, voter information and previous hash. Therefore, ‘*None*’ objects are assigned in their place.

3) **getLastBlock() method:** returns the last element in the 'chain' attribute of the blockchain.

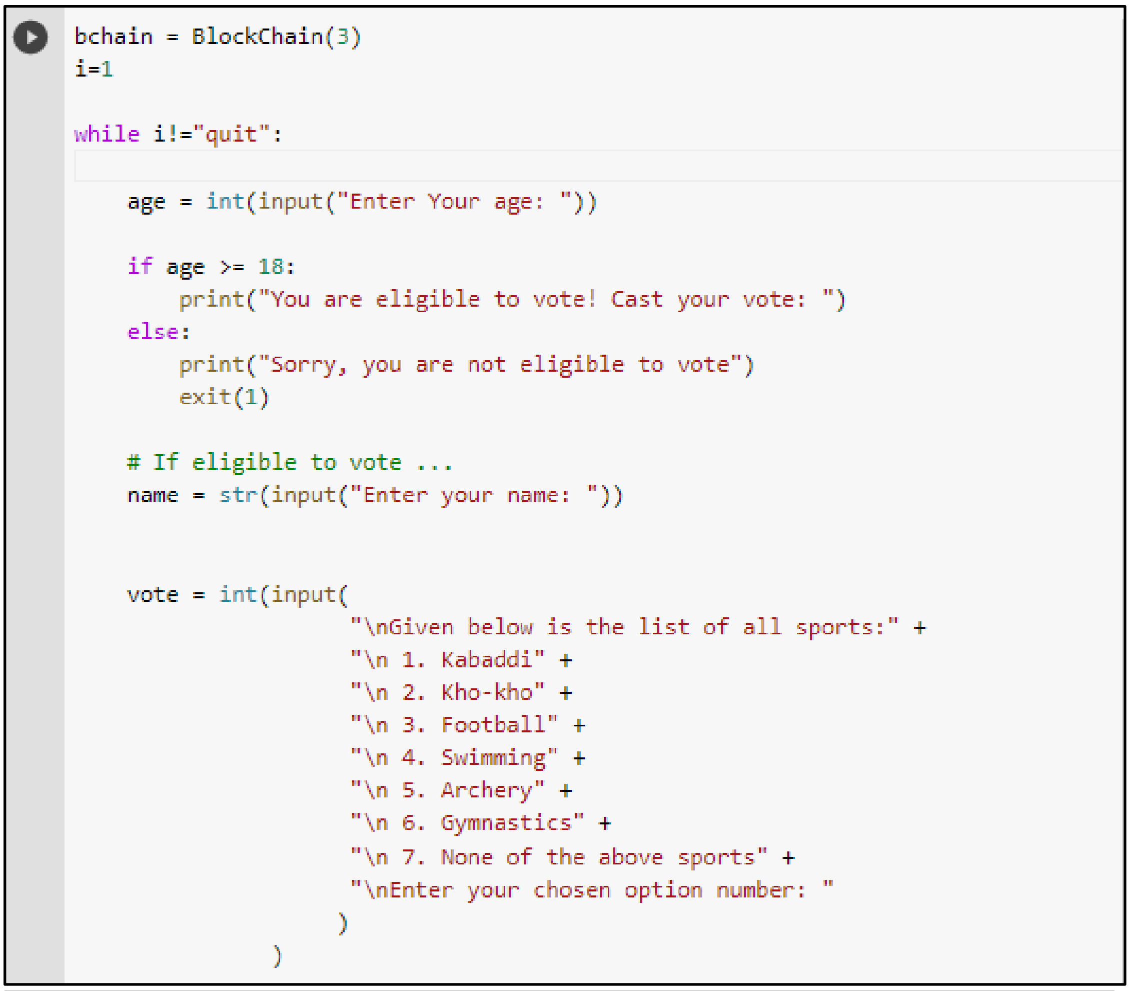
4) **addBlock() method:** The 'previous_hash' attribute of a new incoming block should store the hash of the last block in the existing chain. Nonce and hash that overcome the difficulty level are then 'mined'.

5) **isChainValid() method:** This method checks for the following conditions:

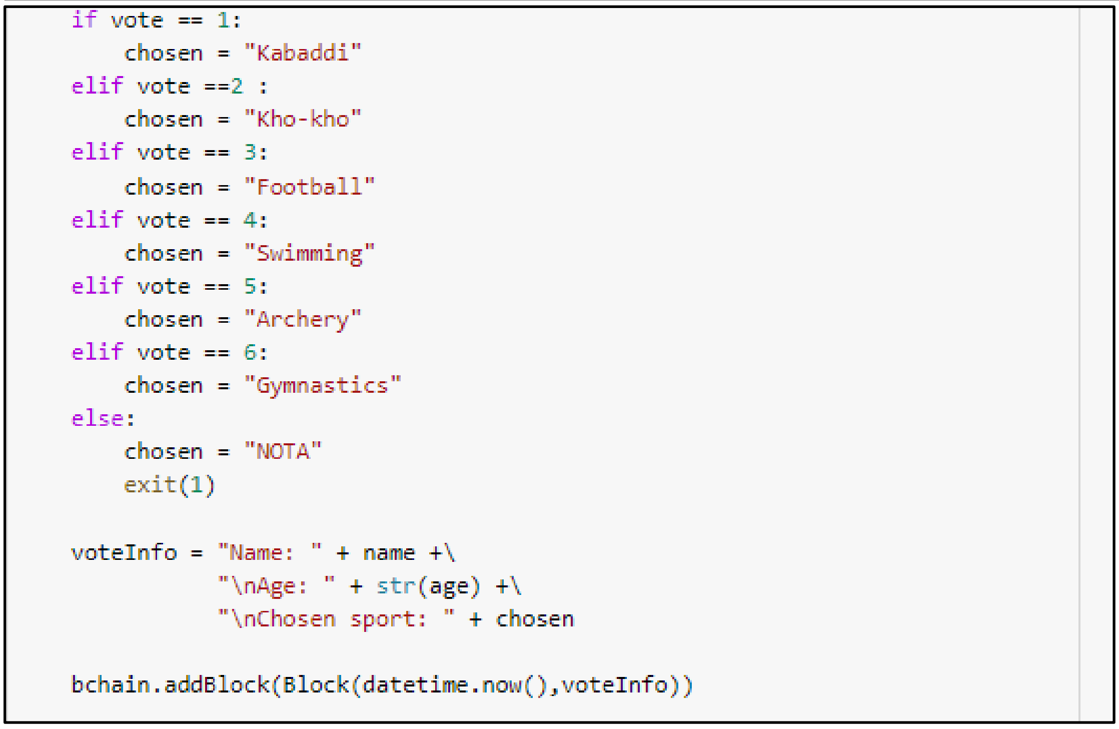
    1) Whether the hash stored is the block's actual hash
    2) The 'previous hash' is actually the hash of the previous hash.

Main codescript

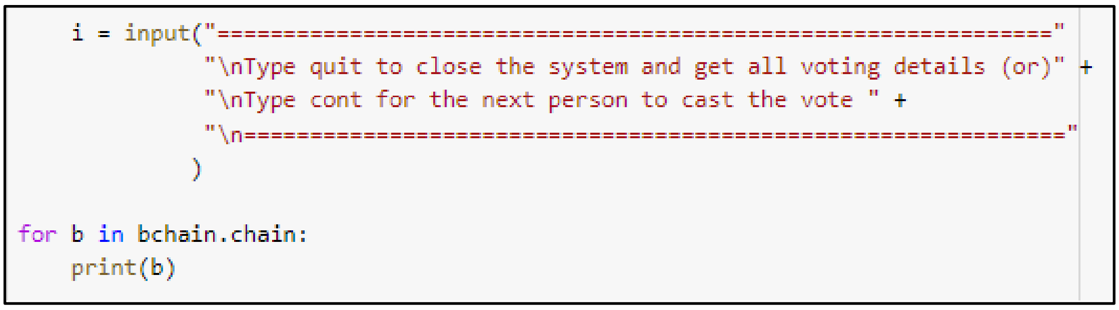
The following code script uses the objects of the classes 'Block' and 'Blockchain' to create a blockchain-based voting system. It does so using an infinite while loop. In every iteration, the user's age is taken as input and voting is allowed if their age is 18 or above. The user can then enter the number corresponding to their chosen sport. If another user is going to cast their vote, then 'cont' needs to be entered. Else, 'quit' can be given and the system stops taking inputs. After closing, the voting details in every block are printed by traversing the list.



```python
bchain = BlockChain(3)
i=1

while i!="quit":

    age = int(input("Enter Your age: "))

    if age >= 18:
        print("You are eligible to vote! Cast your vote: ")
    else:
        print("Sorry, you are not eligible to vote")
        exit(1)

    # If eligible to vote ...
    name = str(input("Enter your name: "))


    vote = int(input(
                     "\nGiven below is the list of all sports:" +
                     "\n 1. Kabaddi" +
                     "\n 2. Kho-kho" +
                     "\n 3. Football" +
                     "\n 4. Swimming" +
                     "\n 5. Archery" +
                     "\n 6. Gymnastics" +
                     "\n 7. None of the above sports" +
                     "\nEnter your chosen option number: "
                    )
               )
```

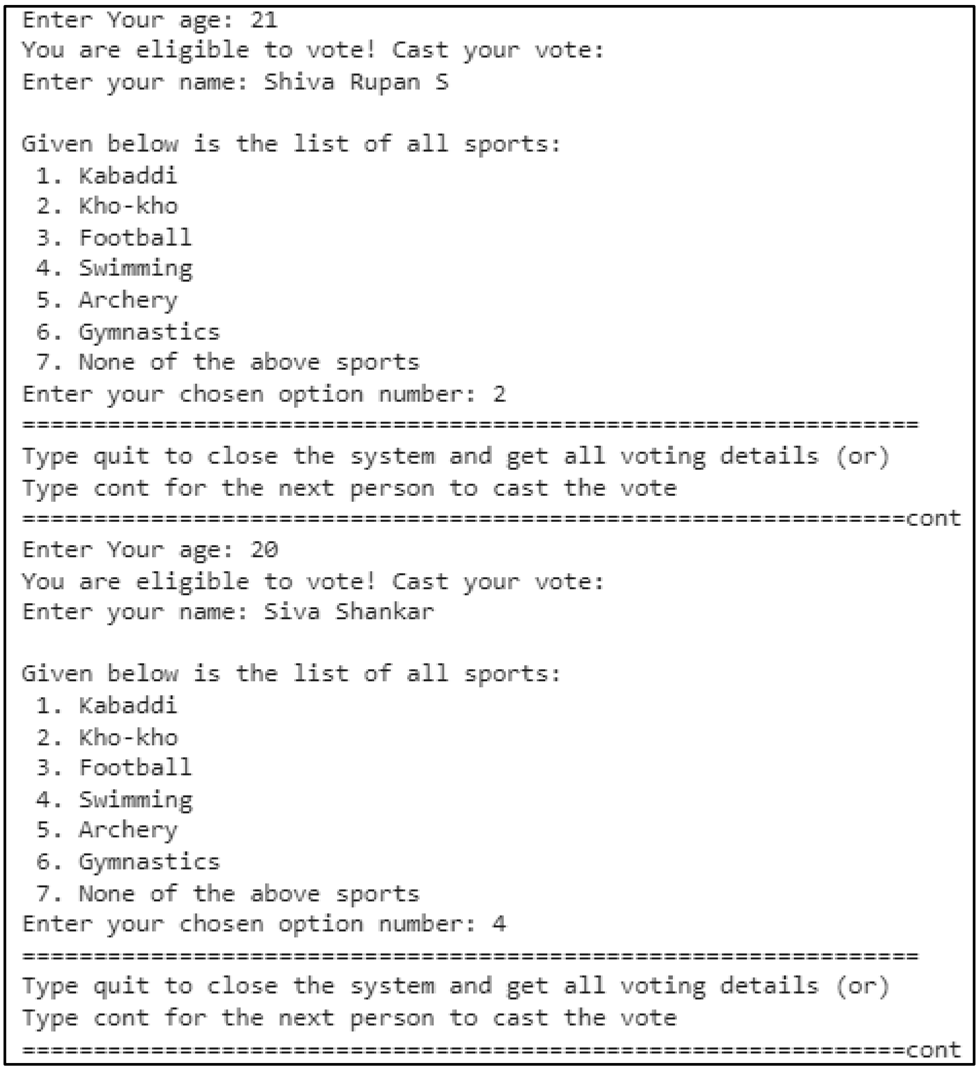


```python
if vote == 1:
    chosen = "Kabaddi"
elif vote ==2 :
    chosen = "Kho-kho"
elif vote == 3:
    chosen = "Football"
elif vote == 4:
    chosen = "Swimming"
elif vote == 5:
    chosen = "Archery"
elif vote == 6:
    chosen = "Gymnastics"
else:
    chosen = "NOTA"
    exit(1)

voteInfo = "Name: " + name +\
           "\nAge: " + str(age) +\
           "\nChosen sport: " + chosen

bchain.addBlock(Block(datetime.now(),voteInfo))
```

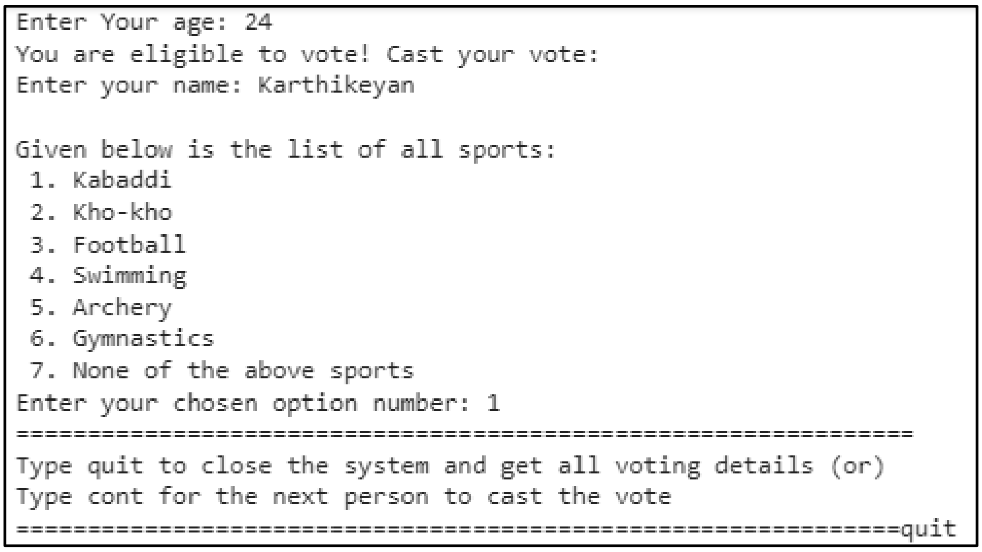


```
    i = input("=============================================================="
              "\nType quit to close the system and get all voting details (or)" +
              "\nType cont for the next person to cast the vote " +
              "\n=============================================================="
             )

for b in bchain.chain:
    print(b)
```

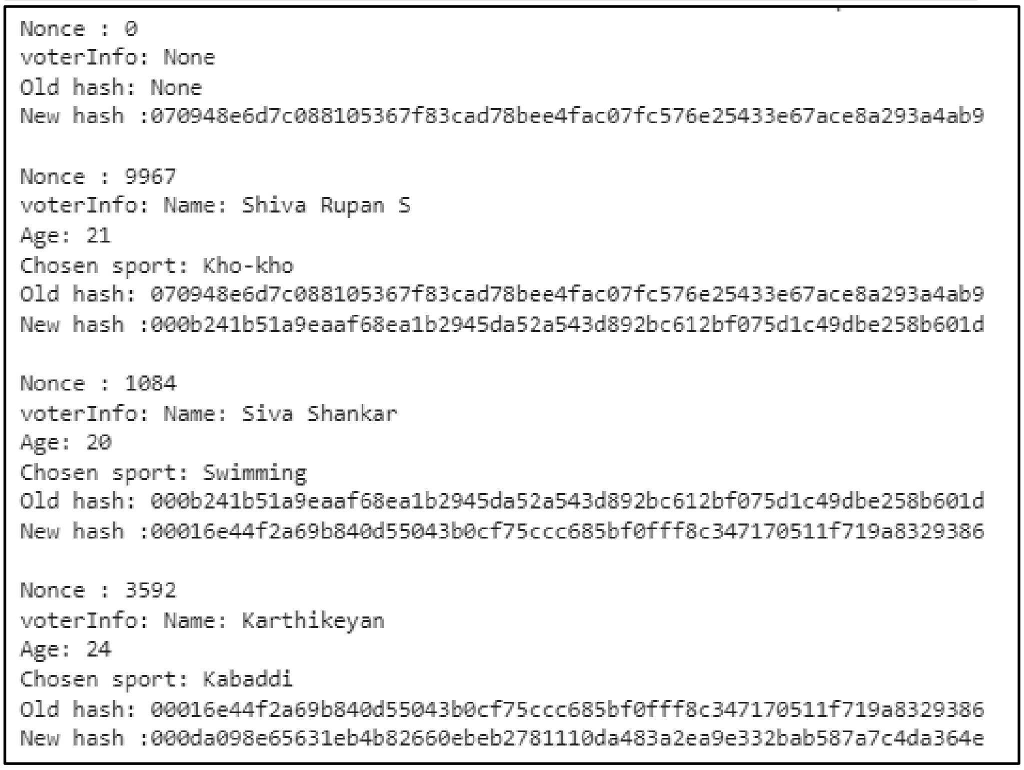
Working of the voting system



```
Enter Your age: 21
You are eligible to vote! Cast your vote:
Enter your name: Shiva Rupan S

Given below is the list of all sports:
 1. Kabaddi
 2. Kho-kho
 3. Football
 4. Swimming
 5. Archery
 6. Gymnastics
 7. None of the above sports
Enter your chosen option number: 2
===============================================================
Type quit to close the system and get all voting details (or)
Type cont for the next person to cast the vote
==============================================================cont
Enter Your age: 20
You are eligible to vote! Cast your vote:
Enter your name: Siva Shankar

Given below is the list of all sports:
 1. Kabaddi
 2. Kho-kho
 3. Football
 4. Swimming
 5. Archery
 6. Gymnastics
 7. None of the above sports
Enter your chosen option number: 4
===============================================================
Type quit to close the system and get all voting details (or)
Type cont for the next person to cast the vote
==============================================================cont
```



```
Enter Your age: 24
You are eligible to vote! Cast your vote:
Enter your name: Karthikeyan

Given below is the list of all sports:
 1. Kabaddi
 2. Kho-kho
 3. Football
 4. Swimming
 5. Archery
 6. Gymnastics
 7. None of the above sports
Enter your chosen option number: 1
===============================================================
Type quit to close the system and get all voting details (or)
Type cont for the next person to cast the vote
==============================================================quit
```



```
Nonce : 0
voterInfo: None
Old hash: None
New hash :070948e6d7c088105367f83cad78bee4fac07fc576e25433e67ace8a293a4ab9

Nonce : 9967
voterInfo: Name: Shiva Rupan S
Age: 21
Chosen sport: Kho-kho
Old hash: 070948e6d7c088105367f83cad78bee4fac07fc576e25433e67ace8a293a4ab9
New hash :000b241b51a9eaaf68ea1b2945da52a543d892bc612bf075d1c49dbe258b601d

Nonce : 1084
voterInfo: Name: Siva Shankar
Age: 20
Chosen sport: Swimming
Old hash: 000b241b51a9eaaf68ea1b2945da52a543d892bc612bf075d1c49dbe258b601d
New hash :00016e44f2a69b840d55043b0cf75ccc685bf0fff8c347170511f719a8329386

Nonce : 3592
voterInfo: Name: Karthikeyan
Age: 24
Chosen sport: Kabaddi
Old hash: 00016e44f2a69b840d55043b0cf75ccc685bf0fff8c347170511f719a8329386
New hash :000da098e65631eb4b82660ebeb2781110da483a2ea9e332bab587a7c4da364e
```

Advantages of using Blockchains

1) **Transparency:** Given that it is a type of a *distributed ledger*, all nodes in the network share a copy of the documentation. The data on a blockchain ledger is easily accessible for everyone to view. If a transaction history changes, everyone in the network can view the change and the updated record. In the case of *crypto currencies*, the transparency of blockchain offers users an opportunity to look through

the history of transactions. Therefore, all information about currency exchange is available for everyone.

2) **Traceability:** In complex supply chains, it is hard to trace products back to their origins. But, with blockchain, the exchanges of goods are recorded so you get an audit trail to learn where a particular asset came from. You also get to know every stop the product made on its journey. This traceability of the products can help verify the *authenticity* and prevent frauds. Blockchain leverages the use of *smart contracts* for employing traceability of assets. So, traceability in blockchain could ensure that any individual could notice the chain of custody and journey of an asset through the supply chain in real-time.

3) **Security:** Since, our data is sensitive and crucial, and blockchain can significantly change how our critical information is viewed**.** By creating a record that can't be altered and is encrypted end-to-end, blockchain helps prevent fraud and unauthorized activity. *Privacy issues* can also be addressed on blockchain by *anonymizing* personal data and using permissions to prevent access. Information is stored across a network of computers rather than a single server, making it difficult for hackers to view data. The shared documentation of transactions can only be updated and/or modified with consensus on a blockchain network. Only if everyone, or a majority of nodes agree to update, the information is edited. Moreover, when a transaction is approved, it is encrypted and connected with the previous transaction. Therefore, no one person or a party has the potential to alter the record.

4) **Increased efficiency and speed:** Traditional paper-heavy processes are time-consuming, prone to human error, and often requires third-party mediation. By streamlining these processes with blockchain, transactions can be completed faster and more efficiently. Documentation can be stored on the blockchain along with transaction details, eliminating the need to exchange paper. There's no need to reconcile multiple ledgers, so clearing and settlement can be much faster.

5) **Automation:** Transactions can even be automated with "*smart contracts*" which increase efficiency and speed the process even further. Once pre-specified conditions are met, the next step in the transaction or process is automatically triggered. Smart contracts reduce human intervention as well as reliance on third parties to verify that terms of a contract have been met. In insurance, for example, once a customer has provided all necessary documentation to file a claim, the claim can automatically be settled and paid.

Drawbacks of using Blockchains

1) **High Energy Consumption:** The blockchain technology gained momentum with the coming of Bitcoin. The *proof-of-work consensus* algorithm explained previously involves a lot of computation. In this process, miners have to find solutions to complex math problems. They receive incentives for completing such tasks. A new incoming transaction means that a new block will be added to the chain. This in turn means that the miners need to find numeric solutions to complex problems at the expense of their computational energy. Nevertheless, there also exist other consensus algorithms that have overcome this problem.

2) **Immutability:** Though use cases like storing *supply chain* information, *transaction records*, etc. leverage on a blochain's *immutability*, this attribute also has a few shortcomings. For instance, if a user reserves tickets via a booking system that runs on blockchain, then cancellation of the ticket will not be possible since it would require that the whole chain be modified.

   Consider another case: If someone wants to restrict the data that they previously made visible to everybody, i.e., want more privacy, then it is virtually impossible to do so.

3) **Storage & speed:** In a blockchain, every node stores a copy of the entire chain. With this, the issue of *storage* arises. Whenever a new block is added to the chain, every node needs to replicate the same in

its copy. In a commercial setup like in a bank, it is essential for the network to be fast and secure at the same time. While *security* is taken care of, speed continues to pose a problem because as the blockchain continues to grow with more nodes and transactions, the whole network is slowed down. This is infeasible since there would be millions of transactions happening every minute or so.

4) **Cost and Implementation:** In spite of blockchain solutions like *Hyperledger* being open source, a lot of investment is needed from the organization that is willing to leverage on it. Apart from hiring blockchain developers who excel at different aspects of blockchain technology, an organization needs to spend a lot on maintenance. For enterprise level projects, the funds required can even shoot up to millions.

5) **Private keys:** Decentralization in a blockchain, say used in a financial setup, also implies that users need authorization to access the information shared to them. For this to happen, every user should have their corresponding private key. Private keys are generated during the wallet creation process. Users must ensure that their private keys are kept confidential. If they happen to lose their key, they will be logged out of their wallets and lose access to it permanently. Therefore, completely trusting the users with their private keys without having any backup involves risk. Also, assigning a centralized authority for managing the chain cannot be a solution since it defeats the purpose of having a consensus.

Applications of Blockchains

1) **Financial Exchanges:** Many companies have popped up over the past few years offering decentralized cryptocurrency exchanges. Using blockchain for exchanges allows for faster and less expensive transactions. Moreover, a decentralized exchange doesn't require investors to deposit their assets with the centralized authority, which means they maintain greater control and security. While blockchain-

based exchanges primarily deal in cryptocurrency, the concept could be applied to more traditional investments as well.

2) **Secure Personal Information:** Keeping data such as your Social Security number, date of birth, and other identifying information on a public ledger (e.g., a blockchain) may actually be more secure than current systems more susceptible to hacks. Blockchain technology can be used to secure access to identifying information while improving access for those who need it in industries such as travel, healthcare, finance, and education.

3) **Securely share medical information:** Keeping medical records on a blockchain can allow doctors and medical professionals to obtain accurate and up-to-date information on their patients. That can ensure that patients seeing multiple doctors get the best care possible. It can also speed up the system for pulling medical records, allowing for more timely treatment in some cases. And, if insurance information is held in the database, doctors can easily verify whether a patient is insured and their treatment is covered.

4) **Non-Fungible Tokens:** Blockchain prevents data from existing in two places, putting an NFT on the blockchain guarantees that only a single copy of a piece of digital art exists. That can make it like investing in physical art but without the drawbacks of storage and maintenance.

5) **Logistics and Supply Chain Tracking:** Using blockchain technology to track items as they move through a logistics or supply chain network can provide several advantages. First of all, it provides greater ease of communication between partners since data is available on a secure public ledger. Second, it provides greater security and data integrity since the data on the blockchain can't be altered. That means logistics and supply chain partners can work together more easily with greater trust that the data they're provided is accurate and up to date.

Conclusion & Future work

Thus, our project has gone from a simple implementation of blockchain to implementing an immutable network of voting information. Thus, we have implemented a simple blockchain-based voting system in Python.

Further, an authentication system involving username and password of users can be implemented in the future. In other words, an end-to-end system for implementing blockchains can be created. ***Private keys*** can be provided to users in order to create a locker-like access to information.

Blockchain technology provides verification efficiencies, including operational, regulatory, enhanced visibility, and traceability. This technology is also a powerful database that could easily be combined with ***big data.*** Blockchain solutions can help cut costs and make many services more competitive.

References

1) Blockchain from Scratch in Python Tutorial: https://www.youtube.com/watch?v=alNU9AVWkQk

2) Github - Eric O Meehan: https://github.com/ericomeehan/sketchbook/tree/python_blockchain

3) Github - rdunlocked18: https://github.com/rdunlocked18/Python-Blockchain-SimpleEvoting

4) Advantages and Disadvantages of Blockchain : https://marutitech.com/benefits-of-blockchain

5) Working of the Blockchain : https://www.youtube.com/watch?v=SSo_EIwHSd4