Programming & Algorithms 2

COURSE WORK TWO - DISTRIBUTED BLOCKCHAIN

AHMED FAROUK MAHMOUD

CU2000512

Introduction and expected learning outcomes

In this course work I was asked to create an application that utilizes a distributed block chain. I had the opportunity to choose any application I wanted, and I had the freedom to choose how to implement my application. Before starting this course work, I did not know much about block chains and how they function. I expect to know how block chain work, how the distributed network functions, and how multithreading can be used in real world applications.

Contents

Application chooses

A blockchain is a decentralized database that is shared among a computer network. A blockchain acts as a database, storing information in a digital format. The blockchain's significance is that it ensures the integrity and security of a data record while also generating trust without the requirement for a trusted third party.

Blockchain vs Database

Features	Blockchain	Database
The data structure	Blocks that form a chain	Tables and columns
Network type	Distributed peer to peer	Centralized server to client
Accessibility	Anyone can access it	Requires permission
Speed	Blockchains are slow	Databases are fast
History	It has a record history and digital record ownership.	It has no history of records or ownership
Confidentiality	Is fully confidential	Is not fully confidential
Operations	Only insert operations	Has create, read, update, and delete
Recursive	Is not recursive. We cannot go back to repeat a task on any record.	Is recursive. We can go back to repeat a task on a particular record.

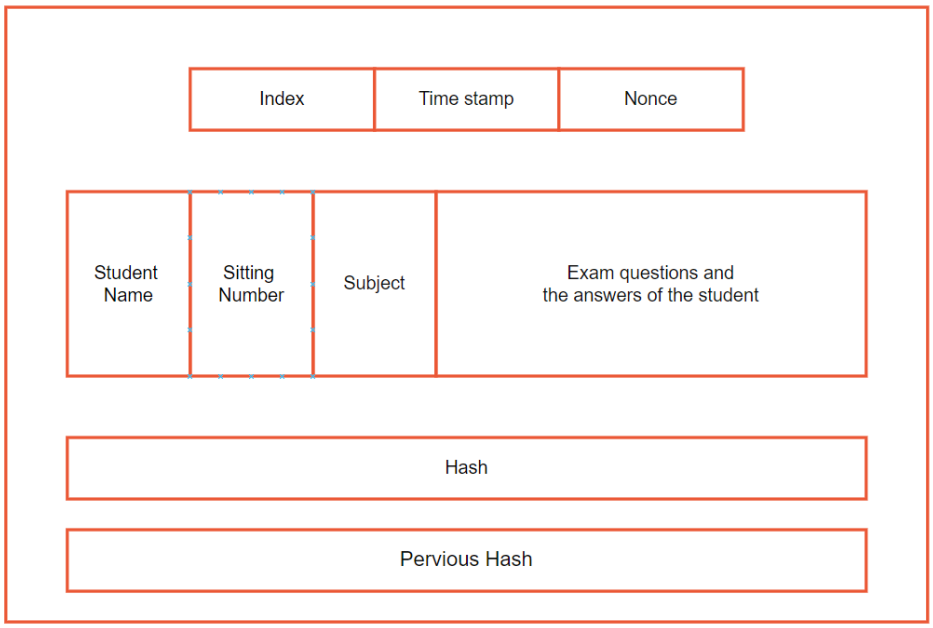
Different application options

In this course, I had the privilege of choosing how to apply my work. A blockchain is applicable in various scenarios, but the priority for me was to build something innovative that would fix a real-world problem that many people face. I once met a friend that was a victim of corruption in the Egyptian scholastic system, Thanaweya Amma. His papers were tampered with, and his grades were lower than they should be. This happens a lot to benefit other students and give them grades they don’t deserve. A lot of students suffer as a result of this type of corruption each year. So, I decided to build a blockchain application to put an end to it.

My application

For the reasons mentioned above I created a distributed blockchain application that can be used as a database to save Thanaweya students exams on, so that the exams can never be tampered with or changed. The application should work in the backend of the software that is used for testing students. Every block on the block chain contains the students name, the students sitting number, the subject, and the answers of the student. The figures below show how a block and the block chain are constructed.

Block diagram



Blockchain diagram

How to use the application

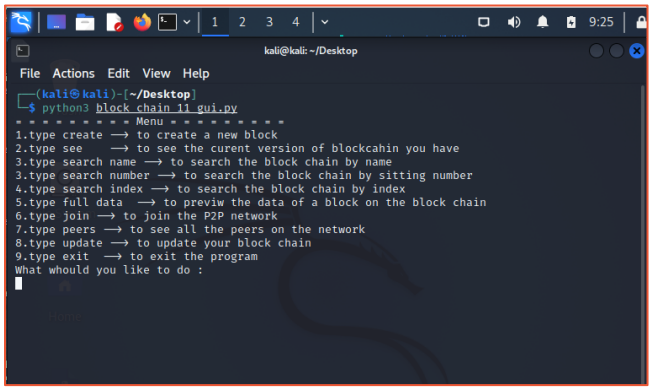
Prerequisites

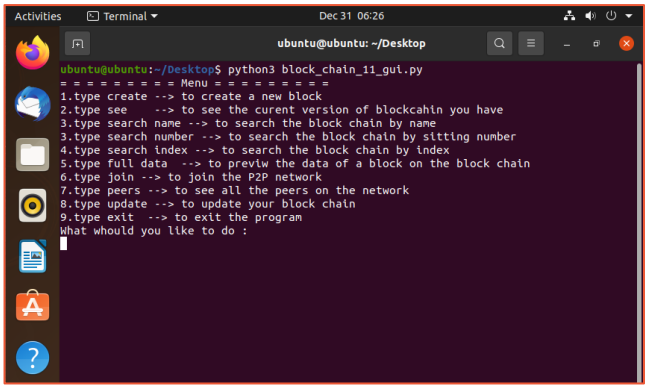
Before starting the program, you need to install the cryptography and Tkinter libraries. Use the below commands to install both libraries.

- Pip Install cryptography
- pip install tk

besides the libraries you must have at least 2 virtual machines working at the same time to create and validate blocks.

Step 1: Start 2 virtual machines and run the python file.





Step 2:

Now both the nodes are not connected. to connect them type join in any of the nodes and type the IP address of the other node.

- Join
- <IP address of the other node> 192.168.247.134

Step 3:

Now that you connected the 2 machines together you can start testing some of the functionalities.

- Type create to create a new block
- Type see to see the current version of the block chain
- Type peers to view all the peers on the network

Note: if you want to connect a third node, just type join on that node and pass the IP address of one of the connected nodes. This will make the third node automatically connect to the first 2 (type peers to verify. If done correctly you will find the 3 IP address representing all the peers on the network)

Table of unite tests

To start testing the functionalities of the program. You must have 2 machines connected.

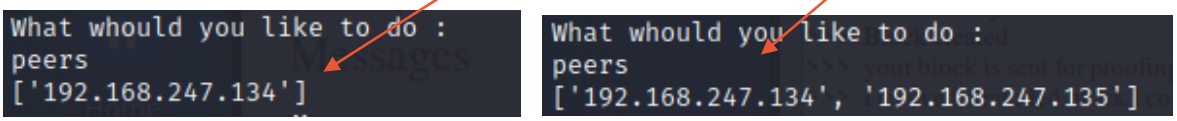
Test Number	Test Name
1	View all peers on the network
2	Connecting a new machine to the network
3	Create a block (Only one node on the network)
4	Create a block (2 nodes on the network)
5	View the current blockchain
6	Search the blockchain by student's name
7	Search the blockchain by student's sitting number
8	Search the blockchain by block's index number
9	Decrypt the data of a block with the private key
10	Graphical user interface testing
11	Trying to connect with an invalid IP address (error handling)
12	Search the blockchain by invalid name (error handling)
13	Search the blockchain by invalid sitting number (error handling)
14	Search the blockchain by invalid index (error handling)
15	Decrypt the data of a block with invalid key (error handling)

View all peers on the network

To view all the peers on the network

- type peers

The result is a list with all the IP addresses of the nodes on the network. The first IP is always the node's IP. If there is only IP visible it means that this node is not connected to the network.



Connecting a new machine to the network

For this test I will assume that we have 2 nodes connected and the third node wants to join the network. To do this all I have to do is type

- Join
- <IP address of the other node> 192.168.247.134

This will result in the new node joining the network. And now if I type peers all the nodes will have 3 IP addresses.

Create a block (Only one node on the network)

Testing if I can create a block while being the only node on the network.

- Type create

```
What whould you like to do :
create
⟶ You are the only node on the network. To create a block there must be atleast 2 active nodes
⟶ to join a node type join then type the ip address of any other node
```

Create a block (2 nodes on the network)

Testing how the block is created, mined, and added to the block chain

- Type create, pass the sitting number, the students name, the subject, and the data

```
What whould you like to do :
create
Please add information to the block :
What is the student's sitting number : 006392
What is the student's name : Ahmed Farouk
What is the subject : Arabic
please atatch the test : --YOUR TEST--
⟶ this is the key of the block do not forget it: Wxl90p3Br0hYJJkEj_sJwv54G5-eJRb9XdPBGSsNs4o=
⟶ Block created
⟶ your block is sent for proofing
```

Step 1

Step 2

After creating the block, the following 4 steps will accrue

1. a message with the generated private key to that block will appear
   (Note: if you do not know this private key you will not be able to later decrypt the block)
2. then the block will be sent to a random node on the network for proofing (Proof of Stake)
3. the random person will proof the new block (mine it) and will added to its block chain then broadcast the newly proofed block
4. after the block is mined and broadcasted it will be received by all the other nodes on the network including the node that created the block and will be added to the block chain.

```
--> I was choosen at random to proof a block. I proofed the block and the block is broadcasted
```

Step 3

```
⟶ I receved a minded block I compared the hash with the previos and added it
```

Step 4

View current version of the blockchain

- Type see (to view the entire block chain)

```
see
{'index': 1, 'time_stamp': '0000-00-00 00:00:00.000000', 'sitting_number': '00', 'student_name': '00', 'su
bject': '00', 'data': '0', 'previous_hash': '00000000000000000000000000000000000000000000000000000000000
000', 'nonce': 20093, 'hash': '000053be8ba01facc38c09bf62dc9e00300adfd7d40702bbbec541f6105bccea'}
{'index': 2, 'time_stamp': '2021-12-31 10:47:04.060517', 'sitting_number': '006392', 'student_name': 'Ahme
d Farouk', 'subject': 'Arabic', 'data': 'gAAAAABhzyX4iT6hz9vildKglqy2sktm4_RoNj9yRhuKYo2WZRCdyGLM-5ya6v0UV
_6WAieBG9qRroLhpgzkbRtpWpDWTn8IPQ==', 'previous_hash': '000053be8ba01facc38c09bf62dc9e00300adfd7d40702bbbe
c541f6105bccea', 'nonce': 49678, 'hash': '0000017026eac26972bb9797238a325ef31db37d5b739e53579c210ec92eeb97
'}
```

```
see
{'index': 1, 'time_stamp': '0000-00-00 00:00:00.000000', 'sitting_number': '00', 'student_name': '00'
, 'subject': '00', 'data': '0', 'previous_hash': '00000000000000000000000000000000000000000000000000
0000000000000', 'nonce': 20093, 'hash': '000053be8ba01facc38c09bf62dc9e00300adfd7d40702bbbec541f6105b
ccea'}
{'index': 2, 'time_stamp': '2021-12-31 10:47:04.060517', 'sitting_number': '006392', 'student_name':
' Farouk', 'subject': 'Arabic', 'data': 'gAAAAABhzyX4iT6hz9vildKglqy2sktm4_RoNj9yRhuKYo2WZRCdyGL
M-5ya6v0UV_6WAieBG9qRroLhpgzkbRtpWpDWTn8IPQ==', 'previous_hash': '000053be8ba01facc38c09bf62dc9e00300
adfd7d40702bbbec541f6105bccea', 'nonce': 49678, 'hash': '0000017026eac26972bb9797238a325ef31db37d5b73
9e53579c210ec92eeb97'}
```

Help

Note: The new block chain is the same for all the peers

Search the blockchain by student's name

to search the block chain by a student name type

- Search name
- Then type the name of the student you are looking for

```
What whould you like to do :
search name
Please enter the name of the student you are looking for: Ahmed Farouk
{'index': 2, 'time_stamp': '2021-12-31 10:10:41.176595', 'sitting_number': '006392', 'student_name': 'Ahme
d Farouk', 'subject': 'Arabic', 'data': 'gAAAAABhz0ehfLBpdy21l6qk-Q5OexUW4Q9bbLbxRV4rK4G33LDYyqYGoWxDwiu02
Rk4iJnE5AJqHfk0luoGUz53dcKuUQVeuw==', 'previous_hash': '000053be8ba01facc38c09bf62dc9e00300adfd7d40702bbbe
c541f6105bccea', 'nonce': 64766, 'hash': '0000aceff1ed8c2f768c7256490667d11884630d30bb5d9111cdd48add7341eb
'}
```

Search the blockchain by student's sitting number

to search the block chain by a student sitting number type

- Search number
- Then type sitting number of the student you are looking for

```
search number
Please enter the sitting number of the student you are looking for: 006392
{'index': 2, 'time_stamp': '2021-12-31 10:10:41.176595', 'sitting_number': '006392', 'student_name': 'Ahmed
Farouk', 'subject': 'Arabic', 'data': 'gAAAAABhz0ehfLBpdy21l6qk-Q5OexUW4Q9bbLbxRV4rK4G33LDYyqYGoWxDwiu02Rk4i
JnE5AJqHfk0luoGUz53dcKuUQVeuw==', 'previous_hash': '000053be8ba01facc38c09bf62dc9e00300adfd7d40702bbbec541f6
105bccea', 'nonce': 64766, 'hash': '0000aceff1ed8c2f768c7256490667d11884630d30bb5d9111cdd48add7341eb'}
```

Search the blockchain by block's index number

to search the block chain by a block's index number type

- Search index
- Then type the blocks index you are looking for

```
What whould you like to do :
search index
Please enter the index of the block you are looking for: 2
{'index': 2, 'time_stamp': '2021-12-31 10:10:41.176595', 'sitting_number': '006392', 'student_name': 'Ahmed
 Farouk', 'subject': 'Arabic', 'data': 'gAAAAABhz0ehfLBpdy21l6qk-Q5OexUW4Q9bbLbxRV4rK4G33LDYyqYGoWxDwiu02Rk
4iJnE5AJqHfk0luoGUz53dcKuUQVeuw==', 'previous_hash': '000053be8ba01facc38c09bf62dc9e00300adfd7d40702bbbec54
1f6105bccea', 'nonce': 64766, 'hash': '0000aceff1ed8c2f768c7256490667d11884630d30bb5d9111cdd48add7341eb'}
```

Decrypt the data of a block with the private key

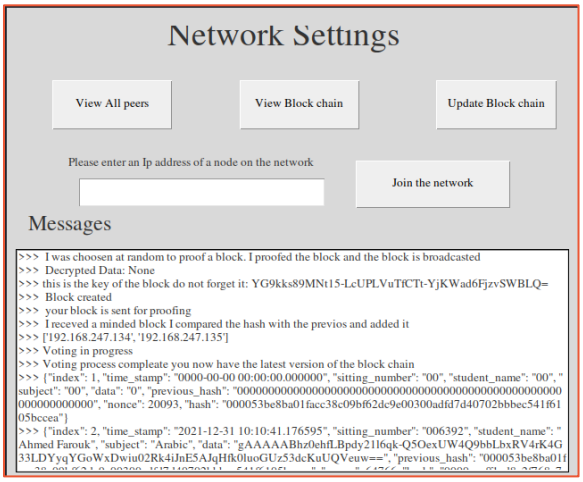
to decrypt a block and get the value of the data inside of it type

- Full data
- Followed by the blocks index
- Then type the private key of the block

```
What whould you like to do :
full data
Please enter the index of the block you are looking for: 2
please enter the key: e3B9Ri3LGdqx09FQktO0cMhaZvp4Qu8zaJzebED8qBs=
Decrypted Data: --YOUR-TEST--
```

Graphical user interface testing

The graphical interface is very simple it is used as a network admin panel. You **cannot** create, search, or decrypt a block with the graphical user interface. You can only view the peers, view the block chain, join the network, and update the block chain.



Trying to connect with an invalid IP address (error handling)

```
What whould you like to do :
join
Please enter an Ip of a node on the network: ahnmed.com
--> Please enter a valied ip of an active node
```

Search the blockchain by invalid name (error handling)

```
What whould you like to do :
search name
Please enter the name of the student you are looking for: mahmoud
⟶  could not find the block associated with name you searched for
None
```

Search the blockchain by invalid sitting number (error handling)

```
What whould you like to do :
search number
Please enter the sitting number of the student you are looking for: 1111
⟶  could not find the block associated with sitting number you searched for
```

Search the blockchain by invalid index (error handling)

```
What whould you like to do :
search index
Please enter the index of the block you are looking for: 56
⟶  index not found on the block chain
```

Decrypt the data of a block with invalid key (error handling)

```
What whould you like to do :
full data
Please enter the index of the block you are looking for: 2
please enter the key: a bad key
⟶  sorry the private key does not match
```

Functionalities

The peer-to-peer network

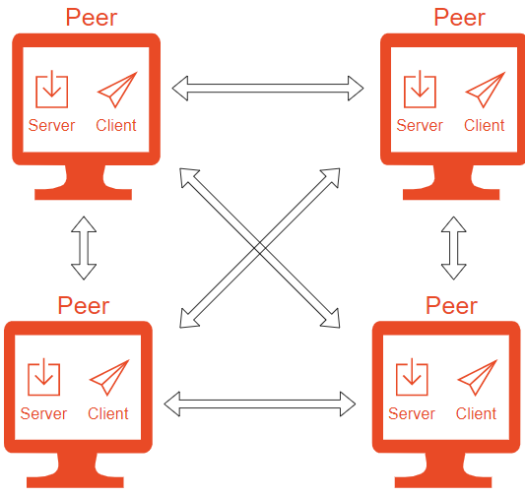
Peer-to-peer networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes. For my application the blockchain is distributed on all the peers on the network.

Server and client sockets

Every peer running the program should have the ability send and receive data simultaneously.

To receive data: every node will bind its IP and static port to a **server socket**

To send data: every node will create a **client socket** and connect it to a peer's IP



Voting function

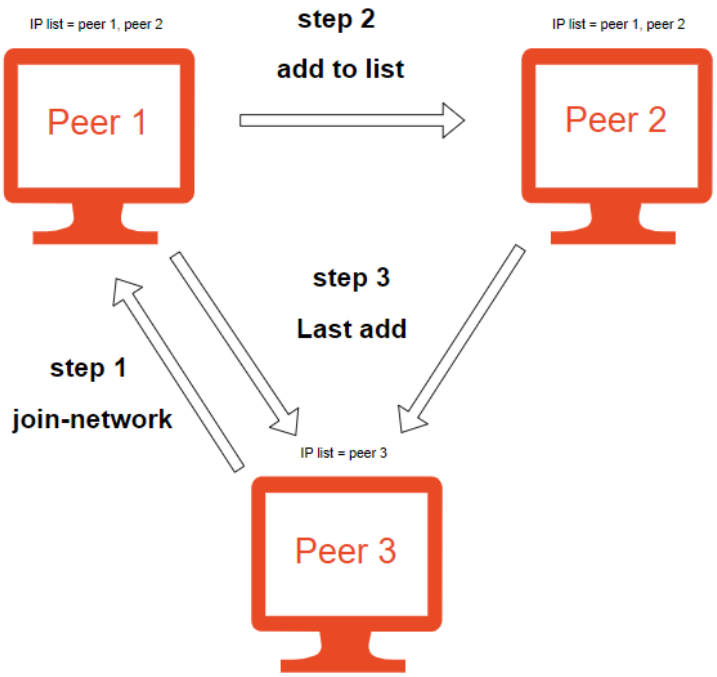
One of the main functions of the program is the voting function. This function makes sure that all the peers on the network have the same block chain. It works by comparing the last block of all the peers and if all the peers have the same block chain, they will send the chain to the peer ruining this function. If they are different, it will vote and take the block chain of the majority. This function automatically runs when a new node joins the network. And to manually run this function type update. This way of voting introduces a new attack named the 51% attack which means that if 51% of the network agree that a wrong block chain is the real blockchain, they will have the ability to edit the block chain at well. More details in Secure programing principles section.

Auto communication protocol

One of the powerful features in my application is the auto communication protocol. This protocol allows for new nodes to automatically connect to the network. Before implementing this functionality, I had to manually add all the IPs of the nodes in the IP list of each node statically. But after implementing this protocol I am now able to connect any new node by just passing only one IP of any node on the network. The protocol works by utilizing 3 different functions.

1. Join-network
2. Add-to-list
3. Last-add

The below diagram explains how the protocol I created works. only peers 2 and 3 are on the network, and peer 3 want to join the network. For peer 3 to join the following 4 steps will happen.



Step 1: Peer 3 will send join-network to peer 1.

Step 2: Peer 1 will receive the join-network command and will add peer 3 IP to its list.

Step 3: Peer 1 will then send a add to list command to peer 2 and this makes peer 2 add peer 3 IP address to its list.

Step 4: finally, Peer 1 and peer 2 will send their IP to peer 3 as a last add message.

Note: at the end all the peers will have all the 3 IP addresses of all the nodes on the network.

Extra Functionalities

Besides the network functionalities, I also added some extra functionalities to the program:

1. Binary search algorithms

   I used a binary search algorithm for the index search. This helps with the efficiency of the search as I can search with O(log N) instead of O(N).

2. Queues

   I used a linked list with a head and tail to implement queues. all the blocks received are enqueued and then dequeued by the appropriate functions. This makes sure that no blocks are dropped even if a node is overwhelmed by blocks.

3. Symmetric Private key encryption

   To ensure confidentiality, I used symmetric key encryption to encrypt all data inside the blocks this is used so that only the student with private key can decrypt the block.

4. Mining and Proof of stake

Data structures and efficiency

In this part I will list all the data structures used in my application. And the reasons why I used every data structure.

Number	Data type	Data structure
1	Block	Custom block class and dictionary
2	Block chain	List
3	Queue	Linked list

1. Block

   The block data type is a custom class with the main data structure being a dictionary. Since I used sockets to create the peer-to-peer network I had to use dictionaries to be able to dump the dictionary into a json file that can then be encoded and send to other peers. The efficiency of the searching inside the dictionary is O(N).

2. Block chain

   The block chain's data structure is a dynamic array (list). While choosing this data structure I had 2 options to choose from a linked list or a dynamic array. Because both do not have a fixed size. The reason I choose a dynamic array over a linked list is that a list uses less memory and can allow for a binary search while the linked list uses more memory, and I cannot implement a binary search. The efficiency of this data structure is inserting and searching by name or sitting number is O(N) but since I use a dynamic array, I was able to search for a block's index with a binary search with an efficiency of O (log N).

3. Queue

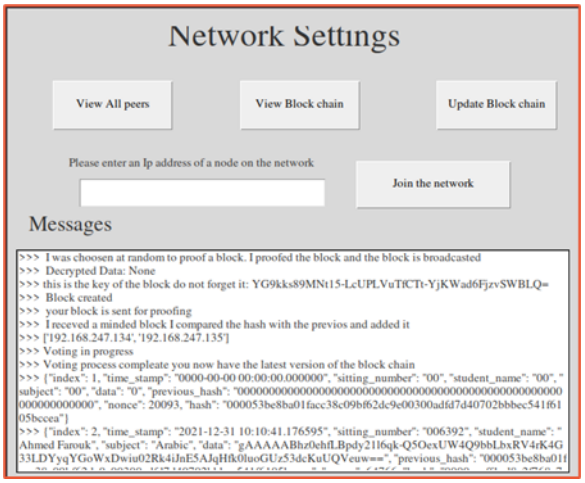
   Like the block chain data type I also had 2 option to choose from while implementing the queue. A dynamic array or a linked list. I used a head and tail linked list because although it uses more memory it allowed me to enqueue and dequeue with O(1) instead of O(N) this is because there is a head and a tail in the linked list pointing at the beginning and the end of the linked list.

User interface

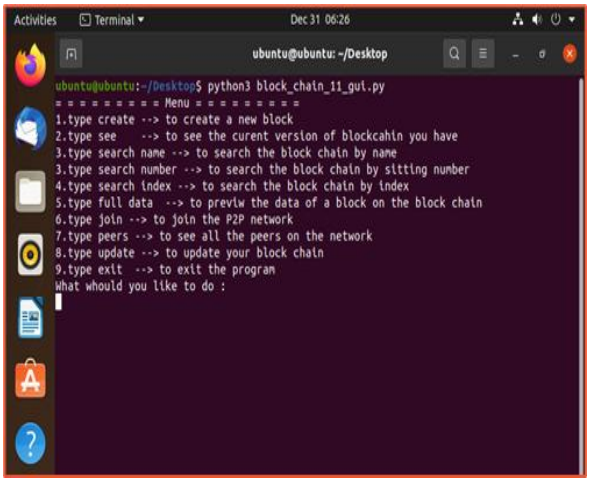
The user interface is how a user interacts with a program. As mentioned earlier in my application section, I choose an application that should run in the back end of another application. With that in mind, my application has 2 user interface options. Option one command line interface(CLI) and option two graphical user interface(GUI). The CLI is used to access all the functionalities of the program while the GUI is used as a network panel only. The below table shows a detailed comparison between the CLI and GUI.

Features	Command line interface	Graphical user interface
View all peers on the network	✓	✓
View entire blockchain	✓	✓
Update block chain	✓	✓
Join a network	✓	✓
Create a block	✓	✗
Search by name	✓	✗
Search by setting number	✓	✗
Search by block index (binary search)	✓	✗
Decrypt a block with a private key	✓	✗

Graphical user interface (network panel only)



Command line interface (all functionalities)

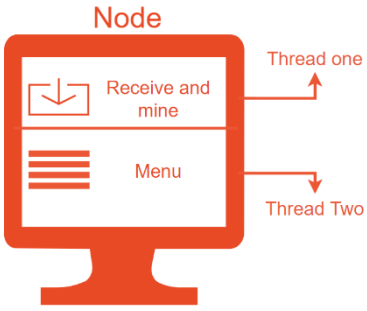


Creating the GUI: I was able to create the GUI with the built-in library of python named Tkinter. I used 4 main widgets: label, button, entry, and frame.

Creating the CLI: I was able to create the CLI with a while loop that takes input from the user and based on the input it would execute different functions.

Threading and Process Synchronization

As explained above in the peer-to-peer section, every node running the program must have the ability to receive, mine, and send blocks simultaneously. To achieve this, I used Multithreading. Multithreading is the ability of a central processing unit to provide multiple threads of execution concurrently, supported by the operating system. In my example, there are 2 main threads running at the same time. One for receiving and mining blocks and another for the menu.



Process Synchronization is a way to coordinate processes that use shared data among different end devices. I used this in my application since the block chain is distributed among all the peers on the network.

Secure programing principles

Number	**Security principle**	**Notes**
1	Input validations	All the inputs have proper validation
2	Proper error handling	When an error happens in the code the code will not crash
3	Closing the connection of every client socket	Every client socket after sending the message will close
4	Proof of stake	Proof of stake allows for a random peer on the network to mine the block to ensure the integrity.
5	Voting function	The voting function allows for democratic negotiations between the peers. So, if a rouge nide tries to manipulate the block chain it will be rejected by the other peers.
6	Symmetric key encryption	Ensures confidentiality

Code flow and structure

The code is structured into 6 classes and 2 functions. Every class and function are responsible for a

Number	Section of the code
1	Block and Blockchain classes
2	The peer-to-peer class
3	The queue data structure
4	App class for GUI
5	The receive function
6	The menu function
7	Threading section

The block and the blockchain classes

The block and Block_Chain classes are the backbone of the program. The block class allows me to create a block object by passing sitting number, student name, subject, and data. Then the index, timestamp, previous hash, nonce, and hash are created automatically. The Block_Chain class has a main list were all the blocks are stored and linked. It also has all the searching, hashing, mining, creating, encrypting, decrypting, and validating functions.

Peer to peer class

The peer-to-peer class is responsible for creating and maintaining the peer-to-peer network. The main constructors of this class are

- My IP (responsible for storing the IP of the node)
- My port (responsible for storing the port)
- IP list (a list with all the IPs of the peers on the network)
- Port list (a list with all the ports of the peers on the network)
- A main server socket responsible for binding the node's IP and port to it

Other functions in this class includes the voting, the broadcast, and the proof of stake functions. All these functions are essential for the program to work efficiently.

The queue data structure

As mentioned above I used a head and tail linked list to implement the queue this allowed me to enqueue and dequeue with O(1) instead of the regular O(N). There are 2 classes that I used that helped me create the linked list. First the node class and second the queue class. The node class creates node

objects that need a value and a next_node. the queue class keeps track of the head and tail of the linked list along with the main functionalities like enqueue and dequeue.

The receiving function

The receiving function is a primary function in my program. It works by first listing on the server socket and awaits connections. When it receives a block, it will enqueue it and if it receives a string will take the appropriate action. If it needs mining, it will mine it and if it needs to be added to the block chain, it will add it and after the operation is complete it will dequeue the block from the queue.

Treading section

This part of the code is the easiest yet very important for the program to work. I call the thread function form the threading library and give it a target to function to run. Then call the start function to start the thread.

GitHub repo

https://github.com/ahmedfarou22/P2P-Python-Blockchain

Source Code

```
import hashlib, datetime, json, socket, threading, random, time # built in
libraries needed for this program to work
from cryptography.fernet import Fernet # used for symmetric encryption


# for gui #######
from tkinter.constants import END, TRUE
import tkinter as tk
import tkinter.font as tkFont

############## Block, BlockChain, And P2P Classes ##############


class App:
    def __init__(self, root):
        #setting title
        root.title("undefined")
        #setting window size
        width=640
        height=526
        screenwidth = root.winfo_screenwidth()
        screenheight = root.winfo_screenheight()
        alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2,
(screenheight - height) / 2)
        root.geometry(alignstr)
        root.resizable(width=False, height=False)
        self.textBox = tk.Text(root)
        self.textBox["borderwidth"] = "1px"
        ft = tkFont.Font(family="Times", size=10)
        self.textBox["font"] = ft
        self.textBox["fg"] = "#333333"
        self.textBox.place(x=10,y=270,width=620,height=250)
        self.textBox.tag_config("warning", background="#fffa65",
selectbackground="black")
        self.textBox.tag_config("error", background="#e74c3c",
selectbackground="black")
        self.textBox.tag_config("control", background="#2ecc71",
selectbackground="black")
        self.textBox.tag_config("basic", background="white",
selectbackground="black")

        GLabel_549=tk.Label(root)
```

```
        ft = tkFont.Font(family='Times',size=28)
        GLabel_549["font"] = ft
        GLabel_549["fg"] = "#333333"
        GLabel_549["justify"] = "center"
        GLabel_549["text"] = "Network Settings"
        GLabel_549.place(x=170,y=20,width=281,height=30)

        GButton_499=tk.Button(root)
        GButton_499["bg"] = "#efefef"
        ft = tkFont.Font(family='Times',size=10)
        GButton_499["font"] = ft
        GButton_499["fg"] = "#000000"
        GButton_499["justify"] = "center"
        GButton_499["text"] = "View All peers"
        GButton_499.place(x=50,y=80,width=136,height=57)
        GButton_499["command"] = self.GButton_499_command

        GButton_575=tk.Button(root)
        GButton_575["bg"] = "#efefef"
        ft = tkFont.Font(family='Times',size=10)
        GButton_575["font"] = ft
        GButton_575["fg"] = "#000000"
        GButton_575["justify"] = "center"
        GButton_575["text"] = "View Block chain"
        GButton_575.place(x=260,y=80,width=136,height=57)
        GButton_575["command"] = self.GButton_575_command

        GButton_363=tk.Button(root)
        GButton_363["bg"] = "#efefef"
        ft = tkFont.Font(family='Times',size=10)
        GButton_363["font"] = ft
        GButton_363["fg"] = "#000000"
        GButton_363["justify"] = "center"
        GButton_363["text"] = "Update Block chain"
        GButton_363.place(x=480,y=80,width=137,height=57)
        GButton_363["command"] = self.GButton_363_command

        self.GLineEdit_248=tk.Entry(root)
        self.GLineEdit_248["borderwidth"] = "1px"
        self.ft = tkFont.Font(family='Times',size=10)
        self.GLineEdit_248["font"] = ft
        self.GLineEdit_248["fg"] = "#333333"
        self.GLineEdit_248["justify"] = "left"
```

```
        self.GLineEdit_248["text"] = "Entry"
        self.GLineEdit_248.place(x=80,y=190,width=277,height=33)

        GButton_175=tk.Button(root)
        GButton_175["bg"] = "#efefef"
        ft = tkFont.Font(family='Times',size=10)
        GButton_175["font"] = ft
        GButton_175["fg"] = "#000000"
        GButton_175["justify"] = "center"
        GButton_175["text"] = "Join the network"
        GButton_175.place(x=390,y=170,width=175,height=55)
        GButton_175["command"] = self.GButton_175_command

        GLabel_470=tk.Label(root)
        ft = tkFont.Font(family='Times',size=10)
        GLabel_470["font"] = ft
        GLabel_470["fg"] = "#333333"
        GLabel_470["justify"] = "center"
        GLabel_470["text"] = "Please enter an Ip address of a node on the
network"
        GLabel_470.place(x=30,y=160,width=354,height=30)

        GLabel_393=tk.Label(root)
        ft = tkFont.Font(family='Times',size=18)
        GLabel_393["font"] = ft
        GLabel_393["fg"] = "#333333"
        GLabel_393["justify"] = "center"
        GLabel_393["text"] = "Messages"
        GLabel_393.place(x=10,y=230,width=122,height=30)


    def print_to_GUI(self, msg, type="basic"):
        self.textBox.configure(state="normal")
        autoscroll = False
        if self.textBox.yview()[1] == 1:
            autoscroll = True
        self.textBox.insert(END, ">>> " +  str(msg) + "\n", str(type) + "\n")
        if autoscroll:
            self.textBox.see(END)
        self.textBox.configure(state="disabled")

    def GButton_499_command(self): # view all peers
        self.print_to_GUI(network.ip_list)
```

```
    def GButton_575_command(self): # view block chain
        for i in range(len(b1.Chain)):
            temp = json.dumps(b1.Chain[i].block)
            self.print_to_GUI(temp)


    def GButton_363_command(self): #Update Block chain
        network.send_random_string("send latest")
        self.print_to_GUI("Voting in progress")
        self.print_to_GUI("Voting process compleate you now have the latest
version of the block chain")



    def GButton_175_command(self): # Join the network
        gui_ip = self.GLineEdit_248.get()
        if gui_ip == "":
            self.print_to_GUI("you did not enter an IP")

        else:
            try:
                if len(network.ip_list) == 1:
                    ip_input = gui_ip
                    j_send = ("j--network" + network.my_ip)
                    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                    client.connect((ip_input,6666))
                    json_block_dict = json.dumps(j_send)
                    client.send(json_block_dict.encode('utf-8'))
                    client.close()
                    time.sleep(2) # Sleep for 2 seconds before geting last
version of the block cahin
                    network.send_random_string("send latest")
                if len(network.ip_list) > 1:
                    self.print_to_GUI(" you are now on the network")
            except:
                self.print_to_GUI('Please enter an ip of a active node other than
your node')


class Block:
    """This class creates block objects evey block contains 9 fields"""
```

```
    def
__init__(self,index,time_stamp,sitting_number,student_name,subject,data,previous_
hash) -> None:
        self.index = index
        self.time_stamp = time_stamp
        self.sitting_number = sitting_number
        self.student_name = student_name
        self.subject = subject
        self.data = data
        self.previous_hash = previous_hash

        self.block = {
            "index":int(self.index), # index used for searching and makes sure
evey block has a diffrent hash
            "time_stamp":str(self.time_stamp), # takes in the time the block was
created in
            "sitting_number":str(self.sitting_number), # the sitting number of
the student also used for searching
            "student_name":str(self.student_name), # the name of student
asosiated with this block
            "subject":str(subject), # the subject that the student took

            "data":str(self.data), # the real data of the exam (this part will be
encrypted)

            "previous_hash":str(self.previous_hash), # takes in the hash of the
previous block
            "nonce": 0, # this is used for the minning process and always starts
at one
            "hash":str("") # the hash of the entire fields above
            }

    def __str__(self) -> str: # edited the __str__ function to be able to print
the block
        return str(self.block)  # return the string version of the block

class Block_Chain:
    """This Class has all the main functions to create the block chain"""
    def __init__(self) -> str:
        self.Chain = [] # the main list where i will store all the block objects
        first_block = Block(1,"0000-00-00
00:00:00.000000","00","00","00","0","0000000000000000000000000000000000000000000
0000000000000000000") # created the genices block and passed static values to it
```

```
        first_block.block["hash"] = self.__hash_function__(first_block) # hashed
all the data given above and saved the hash in the hash field of the block object
        self.__mine_block__(first_block) # minned the first block
        self.Chain.append(first_block) # added the first block to the block chain

    """Used to print out the entire block chain"""
    def __str__(self) -> str: # used to be able to print the entire block chain
        for i in range(len(self.Chain)): # loop over all the blocks in the block
chain list
            print(str(self.Chain[i])) # print every block one at a time
        return str("") # since the str function must return a string I returned
an empty string

    """Returns the last block on the block chain"""
    def __last_block__(self) -> Block:
        return self.Chain[-1] # to do this I called the main list at index -1

    """Normal O(N) searching algorithm """
    def get_block_by_name(self,name : str) -> Block: # the function takes the
name of the student as the argument
        for i in range(len(self.Chain)):  #loop over the main list
            if self.Chain[i].block["student_name"] == name: # cheak if the name
supplied matches the name on the block
                return self.Chain[i] #if yes return that block
        print("--> could not find the block associated with name you searched
for") # if it is not found print could not find index


    """Normal O(N) searching algorithm """
    def get_block_by_sitting_number(self,sitting_number : str) -> Block:  # the
function takes the sitting number of the student as the argument
        for i in range(len(self.Chain)): #loop over the main list
            if self.Chain[i].block["sitting_number"] == sitting_number: #cheak if
the sitting number supplied matches the sitting number on the block
                return self.Chain[i] # if yes return that block
        print("--> could not find the block associated with sitting number you
searched for")  # if it is not found print could not find the block

    """Since all the blocks have index and all the blocks are sorted. I used
binary search"""
    def binarySearch(self,arr, l, r, x):
        if r >= l:         # Check base case
            mid = l + (r - l) // 2
```

```
            if arr[mid].block["index"] == x: # If element is present at the
middle itself
                return mid
            elif arr[mid].block["index"] > x: # If element is smaller than mid,
then it can only be present in left subarray
                return self.binarySearch(arr, l, mid-1, x)

            else: # Else the element can only be present
                return self.binarySearch(arr, mid + 1, r, x)  # in right subarray
        else:
            print("--> index not found on the block chain")# Element is not
present in the array
            return "not found"

    """Function that genrates a random key and encrypts data using it -->
(symmetric encryption)"""
    def encrypt_data(self,plain_data):
        key = Fernet.generate_key() # genrate a random key
        x = key.decode() # the key is genrated in bytes format I used the decode
function to store the key in string format in the var X
        print("--> this is the key of the block do not forget it: " + str(x)) #
print the private key to the user
        a1.print_to_GUI("this is the key of the block do not forget it: " +
str(x))
        f = Fernet(key) #
        encrypted_text = f.encrypt(bytes(plain_data, "UTF-8")) # encrypt the
plane data and return the decoded version of it
        return encrypted_text.decode()

    """Function that takes en encrypted data and decrypts it using a private key
entered by the user"""
    def decrypt_data(self,encrypted_data):
        key = input("please enter the key: ") # take the private key from the
user and store it in the var key
        try:
            f = Fernet(key)
            return f.decrypt(bytes(encrypted_data,"UTF-8")).decode() # decrypte
the data using the private key passed above
        except:
            print("--> sorry the private key does not match")

    """This function usesed to hash a block on 2 levels inner hash and outer
hash"""
```

```
    def __hash_function__(self,block) -> hash:
        header = str(block.block["index"]) + str(block.block["time_stamp"])
+str(block.block["sitting_number"]) +str(block.block["student_name"])
+str(block.block["subject"]) + str(block.block["data"])  +
str(block.block["previous_hash"]) + str(block.block["nonce"])
        inner_hash = hashlib.sha256(header.encode()).hexdigest().encode() # got
all the data in string format from the block passed and asigned it to the header
and hashed all of it
        outer_hash = hashlib.sha256(inner_hash).hexdigest() # hashed the inner
hash just to make sure it is uniqe
        return outer_hash # return the outer hash

    """The main function that allows users to create blocks"""
    def __create_block__(self,sitting_number,student_name,subject,data) -> Block:
        # created a new var named new_block and created a block object and passed
all the needed info to the block note: the data section is passed threw the
encryption function first
        new_block =
Block(str(len(self.Chain)+1),datetime.datetime.now(),str(sitting_number),str(stud
ent_name),str(subject),self.encrypt_data(data),self.Chain[-1].block["hash"])

        new_block.block["hash"] = self.__hash_function__(new_block) # add the
hash to the block

        return new_block # return the block with all the info + encrypted data +
it's hash


    """This Function allows for minning minning is the process of making sure
that the block is valied to be appended on the main chain a mined block will have
its hash start with 0000"""
    def __mine_block__(self,block_to_mine) -> Block:
        while block_to_mine.block["hash"][:4] != "0000": # a while loop that
makes allways runs untill the hash start with 0000

            block_to_mine.block["nonce"] += 1 # add one to the nonce
            block_to_mine.block["hash"] =  self.__hash_function__(block_to_mine)
# re hash the new block and cheak again

        return block_to_mine # to break out of the loop above a block will be
hashed then we can return it
```

```
    """Function that addes blocks to the block chain on 2 conditions (must be
mined -0000-) and the previous hash is corect"""
    def __add_block_to_block_chain__(self,block_to_add) -> None:
        block_to_add_hash = block_to_add.block["hash"]
        if block_to_add_hash [:4] == "0000": # condition 1 the block must be
hashed
            if block_to_add.block["previous_hash"] ==
self.__last_block__().block["hash"]: #condition 2 the previous hash of the new
block must be the same as the hash of the last block
                self.Chain.append(block_to_add) # if the 2 conditions are
satisfied append the new block to the chain
            else:
                print("--> the block's previous hash is incorrect or not mined
properly")
                a1.print_to_GUI(" the block's previous hash is incorrect or not
mined properly")

        else:
            print("--> your block is not mined or incorrect")
            a1.print_to_GUI(" your block is not mined or incorrect")


class Peer_To_Peer:
    """
    This class is responsible for the peer to peer conections. the concept
behinde it is that every simple. every node
    runing this file will create
    1. server socket using an ip and a port # Server part
    2. client socket that will connect to a server socket depending on whitch
node it wants to comunicate with
    every node must have (IP, Port --> to create a server) (a list of IP, a list
of ports --> for all the nodes)
    """
    def __init__(self) -> str:
        self.my_ip = self.__get_real_ip__() # a function that get the real ip of
the device
        self.my_port = 6666 # every server socket will use this port

        self.ip_list = [self.my_ip] # the list of all ips on the network
        self.port_list = [self.my_port] # the list of all ports on the network

        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # create
a server socket
```

```
        self.server.bind((self.my_ip, self.my_port)) #bined the real Ip to the
static port
        self.server.listen() # start the lisining function on the server socket


    """A function that get the real Ip of the device by connecting to google and
saving the ip in a var"""
    def __get_real_ip__(self):
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # start a socket
        s.connect(("8.8.8.8", 80)) # connect the socket to google as an http
request
        real_ip = s.getsockname()[0] # save the given real ip in a var named
real_ip
        s.close() # close the connection with google
        return real_ip # return the real ip

    """A very important function that takes in a dictionary and returns a block
object from this dictionary"""

    def dict_to_block(self,dictt) -> Block:
        a_block =
Block(dictt["index"],dictt["time_stamp"],dictt["sitting_number"],dictt["student_n
ame"],dictt["subject"],dictt["data"],dictt["previous_hash"]) # create a block
object and pass the main arguments to it from the dict
        a_block.block["data"] = dictt["data"] # pass the remaining
        a_block.block["nonce"] = dictt["nonce"]
        a_block.block["hash"] = dictt["hash"]
        return a_block # return the block object


    """a dunction that takes one block and sends it to all  nodes on the
network"""
    def broadcast_block(self,block):
        for q in range(1,len(self.ip_list)): # loop over the list of ip starting
at index one becouse index 0 has this nodes ip and we do not want to create a
broadcast storm
            try:
                client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #
create a socket
                client.connect((self.ip_list[q], self.port_list[q])) # connect to
a peer using his ip and port from list of ips and ports
                json_block_dict = json.dumps(block.block) # dump the dictionary
in the block object to a json file
```

```
                client.send(json_block_dict.encode('utf-8')) # encode the json
file and send it
                client.close() # close the connection

            except: # since not all nodes have to be active this can cause an
error
                pass # so if an error happens just keep pass the error


    """sends a random person on the network a string"""
    def send_random_string(self,string):
        for i in range(0,30):
            try:
                client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #
create an
                randomclient = random.randint(1,len(self.ip_list)-1) #should
start at one becouse the random person should never be the same person
                client.connect((self.ip_list[randomclient],
self.port_list[randomclient]))
                json_string = json.dumps(string) # dump the string in json
                client.send(json_string.encode('utf-8')) # encode the json before
sending
                client.close()
                break
            except: # since not all nodes have to be active this can cause an
error
                continue # so if an error happens continue


    """send a block to a random person on the network (used for proof of
stake)"""
    def send_random_block(self,block): # sends a block to random person
        while True:
            try:
                client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                randomclient = random.randint(1,len(self.ip_list)-1) #should
start at one becouse the random person should never be the same person
                client.connect((self.ip_list[randomclient],
self.port_list[randomclient]))
                json_block_dict = json.dumps(block.block)
                client.send(json_block_dict.encode('utf-8'))
                client.close()
                break
```

```
            except:
                continue


class Node(object):
    """a class that creates node objects these objects are then saved in a linked
list to enqueue and dequeue from it"""
    def __init__(self, value: int, next_node: "Node" = None):
        self.value = value # every node has a value
        self.next = next_node # every node also has a .next pointer

    def __repr__(self):
        return "Node <{}>".format(self.value)

class Queue(object):
    """the main linked list that has a head and a tail --> used for O(1) enqueue
and dequeue"""
    def __init__(self):
        self.head = None # start the linked list with no head and no tail
        self.tail = None

    """the enqueue function is used for """
    def enqueue(self, value: int) -> None:
        new_node = Node(value) # create a node object and pass the value in it

        if self.head is None: # if the head is no this means that this is the
first item in the queue so the head and tail will be the same
            self.head = new_node
            self.tail = self.head
            return

        self.tail.next = new_node # else it is no the same change the tai;
        self.tail = new_node

    """Main function that dequeues from the queue"""
    def dequeue(self) -> int:
        try:
            value = self.head.value
            self.head = self.head.next

            if self.head is None:
                self.tail = None
```

```
            return value
        except:
            print("--> deque is empty")
            pass


    def is_empty(self):
        return self.head is None

    def isnotEmpty(self):
        if self.head is not None:
            return True

    """function that gets the first object in the queue"""
    def first(self):
        return self.head.value

############## Initiating 3 Objects ##############

b1 = Block_Chain()
network = Peer_To_Peer()
q1 = Queue()

root = tk.Tk()
a1 = App(root)
############## Main program starts here 2 functions in Threading ##############

def receive():
    while True:
        client, address = network.server.accept()
        somthing_sent = client.recv(1024)
        decoded_somthing = somthing_sent.decode()
        receved = json.loads(decoded_somthing)

        if type(receved) is dict:
            receved_block = network.dict_to_block(receved)
            q1.enqueue(receved_block)



            if q1.isnotEmpty():
                if q1.first().block["hash"][:4] != "0000":
                    minded_block = b1.__mine_block__(q1.first())
```

```
                    if minded_block.block["previous_hash"] == b1.Chain[-
1].block["hash"]:
                        b1.__add_block_to_block_chain__(minded_block)
                        network.broadcast_block(minded_block) #brodcast  the
minded block
                        print("--> I was choosen at random to proof a block. I
proofed the block and the block is broadcasted")
                        a1.print_to_GUI(" I was choosen at random to proof a
block. I proofed the block and the block is broadcasted")
                        q1.dequeue()

                    else:
                        print("--> I was choosen at random to proof the block but
its previous hash does not match my previous hash")
                        a1.print_to_GUI("was choosen at random to proof the block
but its previous hash does not match my previous hash")
                        q1.dequeue()

            if q1.isnotEmpty():
                if q1.first().block["hash"][:4] == "0000":
                    if q1.first().block["previous_hash"] == b1.Chain[-
1].block["hash"]:
                        b1.__add_block_to_block_chain__(q1.first())
                        print("--> I receved a minded block I compared the hash
with the previos and added it")
                        a1.print_to_GUI(" I receved a minded block I compared the
hash with the previos and added it")
                        q1.dequeue()
                    else:
                        print("--> I receved a minded block that does not match")
                        a1.print_to_GUI(" I receved a minded block that does not
match")
                        q1.dequeue()



        if type(receved) is str:
            if receved == "send latest":
                new_list=[]
                ip, _sent_port = address
                index = network.ip_list.index(str(ip))
                for i in range(len(b1.Chain)):
                    block = b1.Chain[i]
```

```
                    dict_from_block = block.block
                    new_list.append(dict_from_block)

                for k in range(1,len(new_list)): # start at one to not send the
first block
                    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                    client.connect((network.ip_list[index],
network.port_list[index]))
                    dictr = json.dumps(new_list[k])
                    client.send(dictr.encode('utf-8'))
                    client.close()

            if receved[:10] == "j--network":
                ip = receved[10:]
                send = ("add" + ip)
                for i in range(1,len(network.ip_list)):
                    try:
                        client = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                        client.connect((network.ip_list[i],
network.port_list[i]))
                        json_sss = json.dumps(send)
                        client.send(json_sss.encode('utf-8'))
                        client.close()
                    except:
                        pass
                if str(ip) not in network.ip_list:
                    network.ip_list.append(str(ip))
                    network.port_list.append(6666)

                lsend = ("l-add" + network.my_ip)
                client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                client.connect((ip,6666))
                json_lsend = json.dumps(lsend)
                client.send(json_lsend.encode('utf-8'))
                client.close()


            if receved[:3] == "add":
                ip = receved[3:]
                if str(ip) not in network.ip_list:
                    network.ip_list.append(str(ip))
                    network.port_list.append(6666)
```

```
                lsend = ("l-add" + network.my_ip)
                client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                client.connect((ip,6666))
                last_add = json.dumps(lsend)
                client.send(last_add.encode('utf-8'))
                client.close()

            if receved[:5] == "l-add":
                ip = receved[5:]
                if str(ip) not in network.ip_list:
                    network.ip_list.append(str(ip))
                    network.port_list.append(6666)

def menu():
    while True:
        print("= = = = = = = = = Menu = = = = = = = = =")
        print("1.type create --> to create a new block")
        print("2.type see    --> to see the curent version of blockcahin you
have")
        print("3.type search name --> to search the block chain by name")
        print("3.type search number --> to search the block chain by sitting
number")
        print("4.type search index --> to search the block chain by index")
        print("5.type full data  --> to previw the data of a block on the block
chain")
        print("6.type join --> to join the P2P network")
        print("7.type peers --> to see all the peers on the network")
        print("8.type update --> to update your block chain")
        print("9.type exit  --> to exit the program")
        inputt = input("What whould you like to do : \n")

        if inputt == "create":
            if len(network.ip_list) == 1:
                print("--> You are the only node on the network. To create a
block there must be atleast 2 active nodes")
                print("--> to join a node type join then type the ip address of
any other node")

            else:
                print("Please add information to the block : ")
                one =  str(input("What is the student's sitting number : "))
                two =  str(input("What is the student's name : "))
```

```
                three =  str(input("What is the subject : "))
                four = str(input("please atatch the test : "))
                created_block = b1.__create_block__(one,two,three,four)

                network.send_random_block(created_block)
                print("--> Block created")
                a1.print_to_GUI(" Block created")
                print("--> your block is sent for proofing")
                a1.print_to_GUI(" your block is sent for proofing")

        elif inputt == "see":
            print(b1)

        if inputt == "search name":
            name_looking_for = str(input("Please enter the name of the student
you are looking for: "))
            print(b1.get_block_by_name(name_looking_for))

        if inputt == "search number":
            setting_number_looking_for = str(input("Please enter the sitting
number of the student you are looking for: "))
            print(b1.get_block_by_sitting_number(setting_number_looking_for))

        if inputt == "search index":
            try:
                index_looking_for = int(input("Please enter the index of the
block you are looking for: "))

                index_of_block_Chain = b1.binarySearch(b1.Chain, 0, len(b1.Chain)
-1 , index_looking_for)
                if index_of_block_Chain != "not found":
                    print(b1.Chain[index_of_block_Chain])

            except:
                print ("--> sorry I did not understad that" )

        if inputt == "full data":
            try:
                index_looking_for = int(input("Please enter the index of the
block you are looking for: "))

                index_of_block_Chain = b1.binarySearch(b1.Chain, 0, len(b1.Chain)
-1 , index_looking_for)
```

```
                if index_of_block_Chain != "not found":
                    blokaya = b1.Chain[index_of_block_Chain]
                    print("Decrypted Data: " +
b1.decrypt_data(blokaya.block["data"]))
                    for_gui = b1.decrypt_data(blokaya.block["data"])
                    a1.print_to_GUI(" Decrypted Data: " + str(for_gui))
            except:
                print("--> sorry I did not understad that")


        if inputt == "join":
            if len(network.ip_list) == 1:
                try:
                    ip_input = str(input("Please enter an Ip of a node on the
network: "))
                    if ip_input == network.ip_list[0]:
                        print("--> Please enter an ip of a active node other than
your node")
                        a1.print_to_GUI("Please enter an ip of a active node
other than your node")

                    else:
                        j_send = ("j--network" + network.my_ip)
                        client = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                        client.connect((ip_input,6666))
                        json_block_dict = json.dumps(j_send)
                        client.send(json_block_dict.encode('utf-8'))
                        client.close()
                        time.sleep(2) # Sleep for 2 seconds before geting last
version of the block cahin
                        network.send_random_string("send latest")
                except:
                    print("--> Please enter a valied ip of an active node")

            if len(network.ip_list) > 1:
                print("--> you are now on the network")

        if inputt == "peers":
            print(network.ip_list)

        if inputt == "update":
            network.send_random_string("send latest")
```

```
        if inputt == "exit":
            break




receive_thread = threading.Thread(target=receive)
receive_thread.start()

menu_thread = threading.Thread(target=menu)
menu_thread.start()


root.mainloop()
```