Topic	File Sharing App - 2	
Class Description	**Students will able to learn File sharing desktop application Students will learn how GUI buttons work based on server and client.**	
Class	**C-209**	
Class time	**45 mins**	
Goal	● Understand about FTP ● Making functions for GUI	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code	
Class structure	**Warm-Up** **Teacher - led Activity 1** **Student - led Activity 1** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **5 mins**

WARM UP SESSION - 10mins

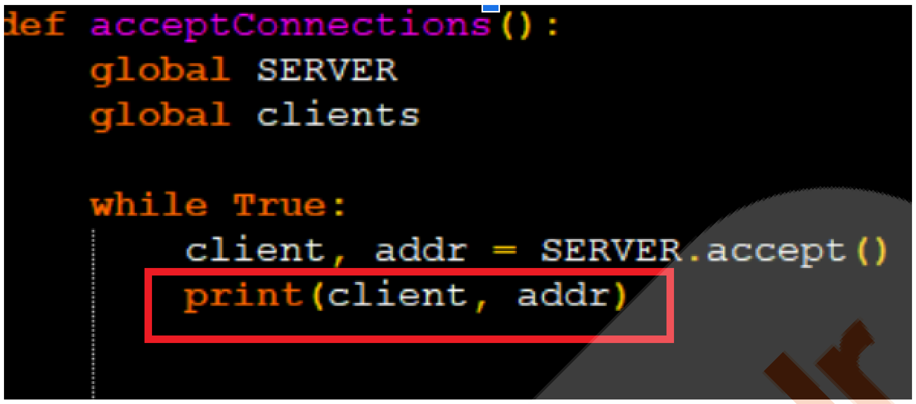
Teacher Action	Student Action
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?	**ESR**: Hi, thanks, yes, I am excited about it!

Q&A Session	
Question	**Answer**
When using the scrollbar() widget, why do we use the xview property? A. To move content from down to up B. To move content from up to down C. To move content from right to left D. None of the above	**C**
Where can we use the scrollbar property? A. Inside listbox and textbox B. Inside listbox only C. Inside textbox only D. None of the above	**A**

TEACHER-LED ACTIVITY - 10mins
Teacher Initiates Screen Share
ACTIVITY • **Write functionality for Connect Buttons** • **Call function at U-I** • **Function for Server side**

Teacher Action	**Student Action**
Okay, so you remember what we did in the last session Great! Any doubts from last session? *The teacher clarifies doubts (if any)*	**ESR** FTP_GUI **ESR:**

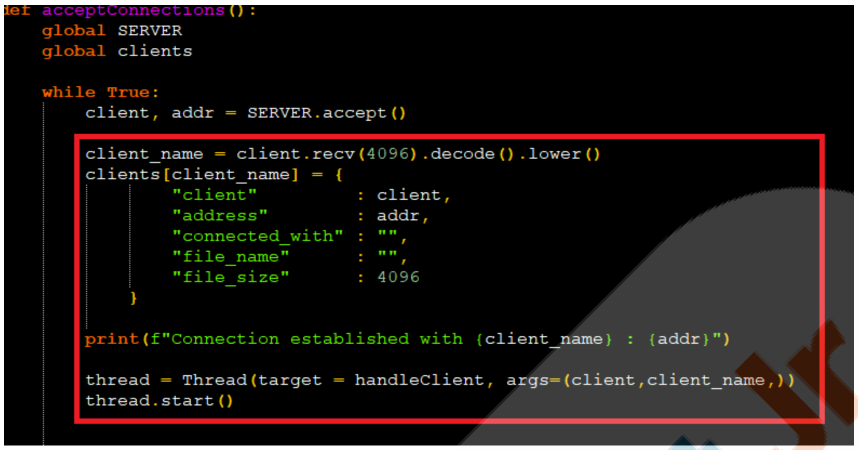
How about moving on to the next part? Let's move on to the second part of the FTP application? You remember what FTP is? The File Transfer Protocol is a standard communication protocol used for the transfer of computer files from a server to a client on a computer network. As we discussed in the last lesson, we set up connections between client and server and designed our FTP user-interface. However, we need functionality at the backend in order to make everything work. Today we will write functions to handle our buttons. Buttons act like events, when we click them, they trigger an output. Today we will cover functionality for connecting to the chat server button, refresh button, connect button, disconnect button. All these buttons need functions at client side and to handle these functions we need to write the code for server side as well.	Yes! **ESR** **Yes!**
Teacher download the boilerplate code from Teacher Activity 1	*Student download the repository from Student Activity 1*
We created an ***acceptConnection()*** method in the previous class, but now we need to store client information. Only the client's name and address were printed in the last session. It looked like this:	


```
def acceptConnections():
    global SERVER
    global clients

    while True:
        client, addr = SERVER.accept()
        print(client, addr)
```

*Let's enhance **acceptConnections()***

Now it displays other details such as the client's address, its file_size, and the file_name it wants to send

- Create the variable client_name where it will store client information that will be received using ***recv()*** ,decode it and then convert it into lower() using ***lower()*** method.
- Create a dictionary where it will store client name, address, connected with information, file name and file size. After getting all the information, display the message in the text area with the client name and address.
- Use threads on the server side so that whenever a client request comes, a separate thread can be assigned for handling each request. It will target the handleclient function and pass two arguments client and client name and use ***start()*** *t*o start this thread process.



```
def acceptConnections():
    global SERVER
    global clients

    while True:
        client, addr = SERVER.accept()

        client_name = client.recv(4096).decode().lower()
        clients[client_name] = {
                "client"         : client,
                "address"        : addr,
                "connected_with" : "",
                "file_name"      : "",
                "file_size"      : 4096
            }

        print(f"Connection established with {client_name} : {addr}")

        thread = Thread(target = handleClient, args=(client,client_name,))
        thread.start()
```

Now we need a way to send messages from the client to the server.

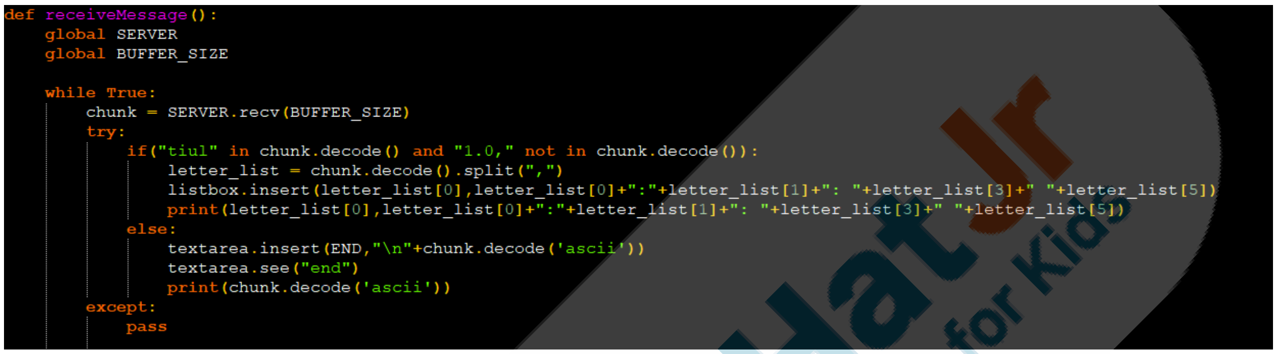
receiveMessage() is a client end function where the message received from a client or server is processed.

- Use global variable SERVER and BUFFER_SIZE
- As the received message is also indefinite, the process will use While loop.
- Create a variable Chunk which will store Buffer size of the data received by server using ***recv()*** function
- If the message contains the strings "tiul" and "1:" the client will understand that this message contains the client data for the first client in the list of clients stored on server and so the client app will remove the old client list from the Active Users List Box with function listbox.delete(0,"end") and will insert the client data of this client in the Listbox. Removing the old data from the list box will avoid duplication
- Else get information from the user list that shows the message to the text area ***.see(end)*** will check if a string is visible within a given range . Print the

same decoded data.

In all except conditions, still we need to write other conditions so let's pass this except condition.

Boiler Code ends here



```python
def receiveMessage():
    global SERVER
    global BUFFER_SIZE

    while True:
        chunk = SERVER.recv(BUFFER_SIZE)
        try:
            if("tiul" in chunk.decode() and "1.0," not in chunk.decode()):
                letter_list = chunk.decode().split(",")
                listbox.insert(letter_list[0],letter_list[0]+":"+letter_list[1]+": "+letter_list[3]+" "+letter_list[5])
                print(letter_list[0],letter_list[0]+":"+letter_list[1]+": "+letter_list[3]+" "+letter_list[5])
            else:
                textarea.insert(END,"\n"+chunk.decode('ascii'))
                textarea.see("end")
                print(chunk.decode('ascii'))
        except:
            pass
```

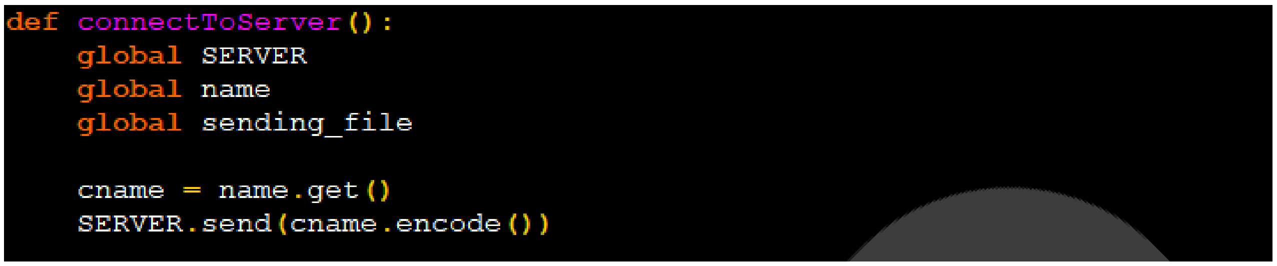
Teacher will start writing code from here

We will start with our first button from the top i.e. ***"connect to chat server"***

By clicking the Connect to chat server button, the client will establish a connection with the server

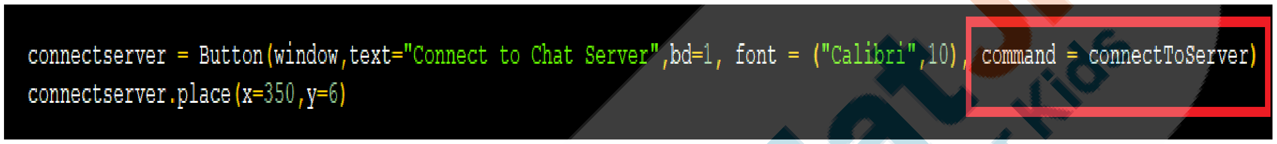
Let's find out how the function name ***ConnectToServer()*** works

- Create global variables SERVER, name, and global sending_file
- To save the name of the client, we'll create a variable cname and use the ***get()*** method to fetch the data from the ***entry()*** widget.
- In order to send username data, the server uses the ***send()*** function and encodes it.



```
def connectToServer():
    global SERVER
    global name
    global sending_file

    cname = name.get()
    SERVER.send(cname.encode())
```

In order to make this button work, let's go to our connectserver at U-I Interface and add an event.



```
connectserver = Button(window,text="Connect to Chat Server",bd=1, font = ("Calibri",10), command = connectToServer)
connectserver.place(x=350,y=6)
```

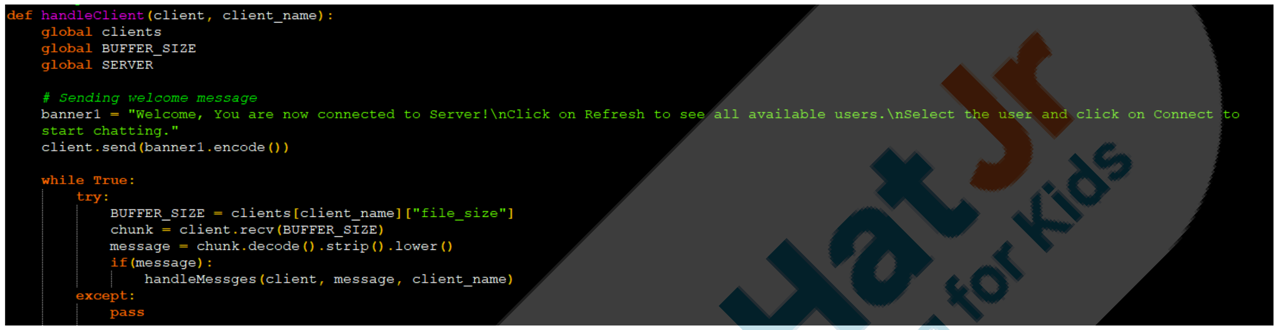
As soon as we connect with the server, we will receive the welcome message in the text area. To display welcome messages in the text area, we must write a function at the server side.

Thus, we created a function ***handleClient()*** where we pass two parameters, client and client_name.

It displays the message according to the client and client request

- We declare global variables client, buffer size, and SERVER. Create a variable banner, where the welcome message will appear.
- Using ***send(),*** the client will send the method while encoding it.
- As this process is indefinite it will use a while loop.
- Buffer_size will store clients which will include client name along with filesize
- Chunk variable will store data that client will receive using ***recv()*** and decode the chunk, and represent in

lower using ***lower()*** method ● If you click a button, the message will display according to the button you clicked. We will make ***handlemessage()*** for the refresh button, connect button, and disconnect button later on.	



```
def handleClient(client, client_name):
    global clients
    global BUFFER_SIZE
    global SERVER

    # Sending welcome message
    banner1 = "Welcome, You are now connected to Server!\nClick on Refresh to see all available users.\nSelect the user and click on Connect to
    start chatting."
    client.send(banner1.encode())

    while True:
        try:
            BUFFER_SIZE = clients[client_name]["file_size"]
            chunk = client.recv(BUFFER_SIZE)
            message = chunk.decode().strip().lower()
            if(message):
                handleMessges(client, message, client_name)
        except:
            pass
```

Teacher Stops Screen Share

STUDENT-LED ACTIVITY - 20 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Fullscreen.**

<u>ACTIVITY</u>

- **Write functionality for Disconnect Button at client side**
- **Write functionality for Disconnect Button at Server side**

Teacher Action	Student Action
Guide the student to get the boilerplate code from [Student Activity 1]()	*Student clones the code from [Student Activity1]()*
Now let's move to second button i.e is Refresh Button We're connected to the server now. Let's move on to the next button, which is **"Refresh"**	

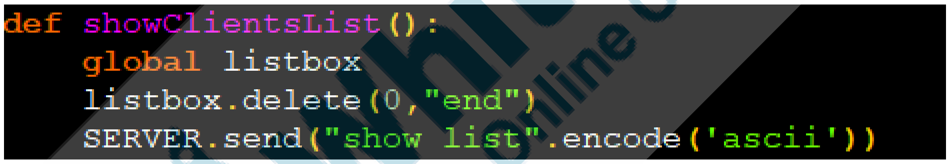
What happens when we click the Refresh button?

Let's get started on our refresh button now.

By clicking the Refresh button, we will see a list of all available online users.

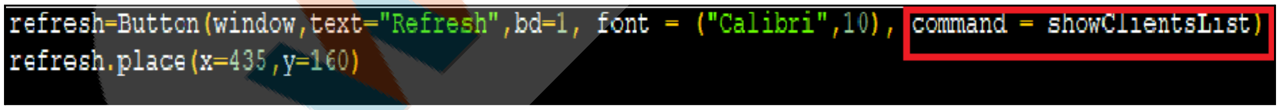
To show all users in the listbox, we will have to create a new function ***showClientList()*** at the client end.

- In order to display all users in the list box, let's create a global variable listbox that can be accessed at any time. All data should be displayed in the listbox from the start to the end, i.e. (0,END).When this is done, send this information to the server using send() in encoded form.



```
def showClientsList():
    global listbox
    listbox.delete(0,"end")
    SERVER.send("show list".encode('ascii'))
```

To make Refresh button working we need to call this function at our user-interface side



```
refresh=Button(window,text="Refresh",bd=1, font = ("Calibri",10), command = showClientsList)
refresh.place(x=435,y=160)
```

In order to handle all requests made by the refresh button at the client side, this function has to be implemented at the server side as well.

Let's name the function ***handlesShowList()***

- function handleShowList is a server-side function that retrieves a list of clients whenever a client requests it.

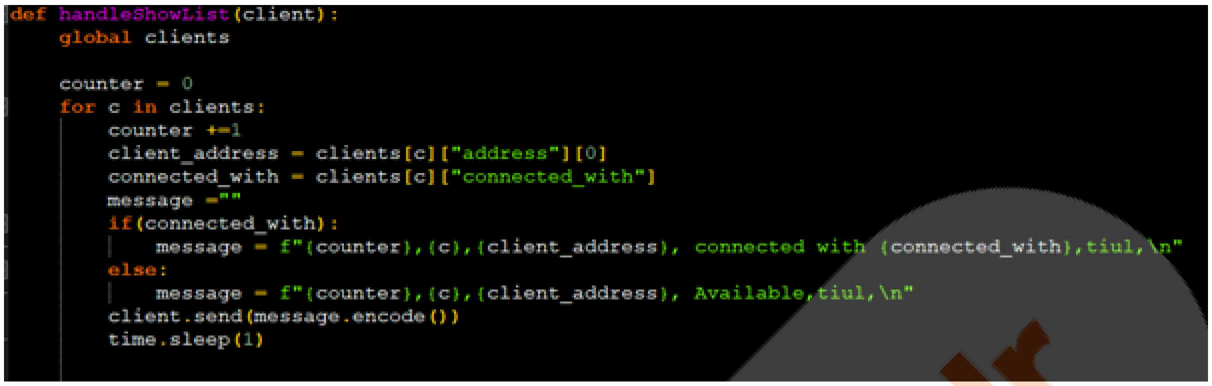
- It takes one argument, client, and clients is a global variable containing the details of all clients connected to the server at the moment.

- Initialize the variable counter to 0 to obtain the number and position of clients in the list

- The for loop with a pointer c is used to fetch clients from a global list of clients. Counters are incremented and set according to the client's position on the list. The variable client_address will contain the IP address of the client while the variable connected_with will contain the name of the client to which it is connected and will be empty if it is not connected to any other client.

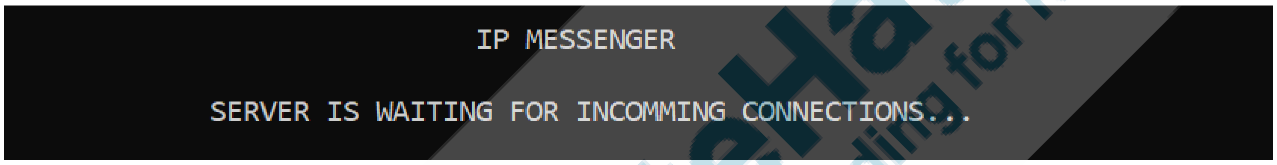
- Once all the above information has been fetched about the client, the message is constructed accordingly to send it to the client window. The name of the client will be stored in variable c.

Note: *An additional text “tiul” has been added here at the end of the message for the client to know that this is user list(tiul). Whenever the client will find this additional text in a message from the server, it will know that this is a user list and has to display this message in the Active Users List Box and not in the Chat Window.*



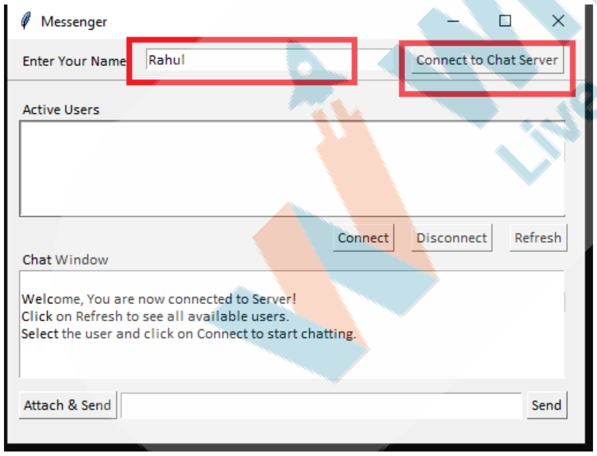
```
def handleShowList(client):
    global clients

    counter = 0
    for c in clients:
        counter +=1
        client_address = clients[c]["address"][0]
        connected_with = clients[c]["connected_with"]
        message =""
        if(connected_with):
            message = f"{counter},{c},{client_address}, connected with {connected_with},tiul,\n"
        else:
            message = f"{counter},{c},{client_address}, Available,tiul,\n"
        client.send(message.encode())
        time.sleep(1)
```

server.py in terminal/cmd looks like -



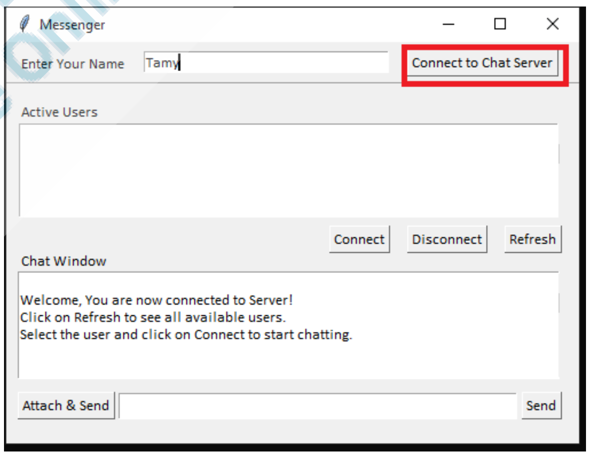
```
                    IP MESSENGER

     SERVER IS WAITING FOR INCOMMING CONNECTIONS...
```

client.py in the terminal/cmd looks like -
Connect two clients by running client.py twice



Rahul

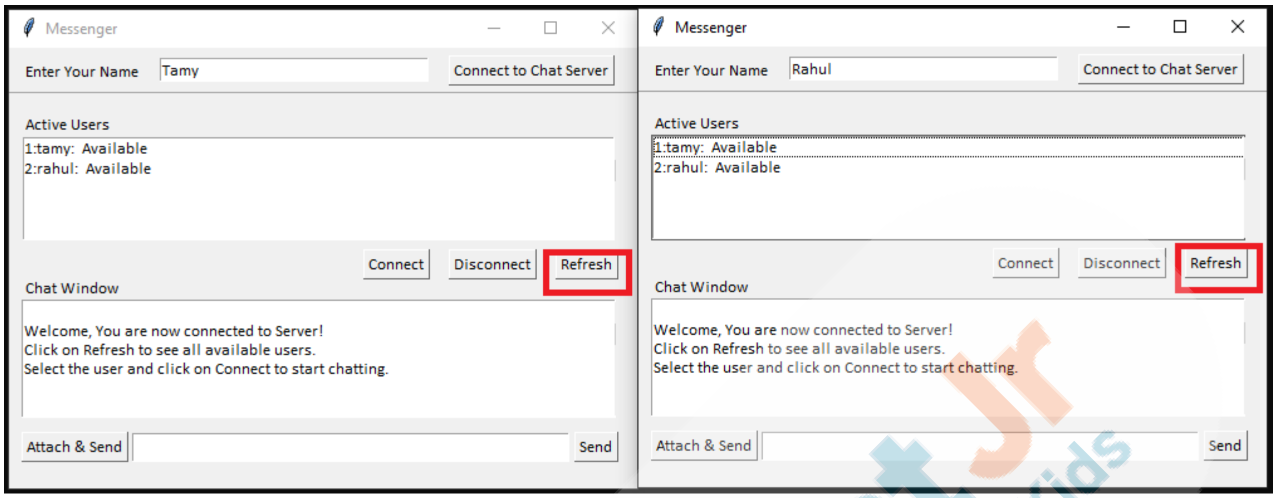
Welcome, You are now connected to Server!
Click on Refresh to see all available users.
Select the user and click on Connect to start chatting.



Tamy

Welcome, You are now connected to Server!
Click on Refresh to see all available users.
Select the user and click on Connect to start chatting.

You need to rerun client.py in order to connect to someone
On clicking refresh button it will display user list, below window will appear like this



Messenger

Enter Your Name: Tamy — Connect to Chat Server

Active Users
```
1:tamy:  Available
2:rahul:  Available
```
Connect — Disconnect — Refresh

Chat Window
```
Welcome, You are now connected to Server!
Click on Refresh to see all available users.
Select the user and click on Connect to start chatting.
```
Attach & Send — Send

Messenger

Enter Your Name: Rahul — Connect to Chat Server

Active Users
```
1:tamy:  Available
2:rahul:  Available
```
Connect — Disconnect — Refresh

Chat Window
```
Welcome, You are now connected to Server!
Click on Refresh to see all available users.
Select the user and click on Connect to start chatting.
```
Attach & Send — Send

Amazing! Our second part is complete! Now, in the next class, we will be working on the function part of the other two buttons

Teacher Guides Student to Stop Screen Share

WRAP UP SESSION - 5 Mins

Quiz time - Click on in-class quiz

Question	**Answer**
What is the purpose of the upper() method? A. Convert upper case string to lowercase() B. Convert to numbers C. Convert lower case string to uppercase() D. None of the above	**C**
What is the purpose of decoding in sockets? A. To convert strings to bytes B. To convert bytes to strings C. Convert string to numbers D. Convert numbers to strings	**B**

WhiteHat Jr
Live online Coding for Kids

What is the use of the append() method? A. Add inside the strings B. Take one string C. Divide Two strings D. Split data into smaller Chunks	**A**
End the quiz panel	
<u>FEEDBACK</u> ● **Appreciate the students for their efforts in the class.** ● **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**	
Teacher Action	**Student Action**
You get Hats off for your excellent work! In the next class we will work on other two buttons	*Make sure you have given at least 2 Hats Off during the class for:* Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
Project Discussion **Goal of the Project:** In Class we created a File Sharing application part two. In class we worked on the user interface buttons.We have written functions for Connect, Disconnect and Refresh buttons for both client and Socket.	

Story:

Maria enjoys listening to music. She gets bored with youtube and other apps. She wishes to create her own music desktop app, so whenever she becomes bored, she can click on her application and listen to a song, download a playlist, or even make a new playlist.Your task is to use Tkinter and write functions for Play and Stop Button

Teacher Clicks ✖ End Class

ADDITIONAL ACTIVITIES

Additional Activities
Encourage the student to write reflection notes in their reflection journal using markdown.

Use these as guiding questions:

- What happened today?
  - Describe what happened.
  - The code I wrote.
- How did I feel after the class?
- What have I learned about programming and developing games?
- What aspects of the class helped me? What did I find difficult?

The student uses the markdown editor to write her/his reflections in the reflection journal.

WhiteHat Jr
Live online Coding for Kids

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity1	Boilerplate Code	https://github.com/pro-whitehatjr/PRO-C209-Teacher-BoilerplateCode
Teacher Activity 2	Reference Code	https://github.com/pro-whitehatjr/PRO-C209-ReferenceCode
Student Activity 1	Boilerplate Code	https://github.com/pro-whitehatjr/PRO-C209-Student-BoilerPlate
In Class Quiz	Class -Quiz	https://s3-whjr-curriculum-uploads.whjr.online/2502b64d-2edd-4f36-a978-00726dec024f.docx