Topic	File Sharing App - 1	
Class Description	**Students will able to learn File sharing desktop application Students will create socket and client connection and will make GUI for Desktop Application**	
Class	**C-208**	
Class time	**45 mins**	
Goal	● Understand about FTP & its Port ● Socket & Client Connection ● User- Interface for desktop Application	
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**

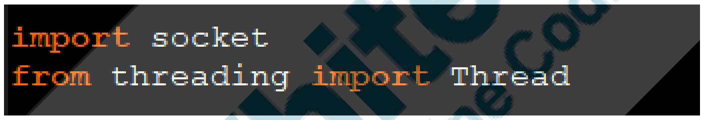
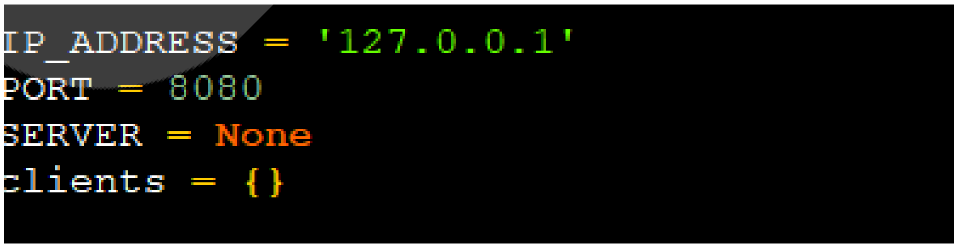
WARM UP SESSION - 10mins

Teacher Action	Student Action

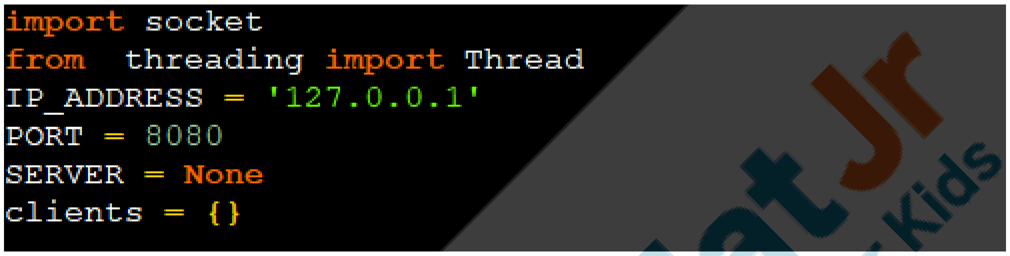
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?	**ESR**: Hi, thanks, yes, I am excited about it!
Q&A Session	
Question	**Answer**
What do you understand about LAN? A. Local Area Network B. Live Area Network C. Locate Area Network D. Levell Area Network	**A**
What is the use of the accept() function in server? A. To accept a connection request from a client B. To accept a connection request from a server C. To reject a connection D. To accept a connection request from database	**A**
TEACHER-LED ACTIVITY - 10mins	
Teacher Initiates Screen Share	
ACTIVITY • **Server & Client Connection** • **GUI For FTP**	
Teacher Action	**Student Action**
So, you remember what we did in last session, Great! Any doubt from the last session? *Teacher resolves the query of the student (if any)*	**ESR:** Varied!

Great! Let's move to a new session? 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 And we discuss Ports too? Can you recall what ports are ? A client program specifies a particular server program on a computer in a network by its connection place Great! So we discussed both ports and FTP and we discussed this too that FTP used port no 21. Right! Don't you want to see a real life application where FTP uses port 21 to transfer files and can access files from your system and transfer it where you want ? What is IP Messenger? Have you heard of it?	**ESR:** **Yes!** **ESR** **Yes!** **ESR**
Open the Ip messenger activity *Teacher Activity 1*	Student will open the *Student Activity 1*
IP Messenger is a desktop application that allows you to chat within your local area network. Using it, clients from different systems can stay in contact with one another and	

can share files with each other. What if we make our own? Let me show how we can make that, To begin with, we will preview what we will be covering in the next four sessions. Wouldn't it be cool if we could create this kind of desktop application? To make such an application first will we note down our requirements? What do you think we need to make such an application? You must be wondering why I am asking this? You know it's going to be a big application so we need planning to do that? All big things need planning right? So for this application we need ● Server ● Client ● GUI ● FTP ● and a lot of functions to make these things happen behind the scenes ? Therefore, we will create things together We know how to make connections between server and	 **ESR** **Yes!**

client as we were doing from the last so many sessions?	
Teacher download the boilerplate code from [Teacher Activity 2]() *Note :- Code to create a server is already provided in the boilerplate code. Explain the code to the Student.*	*Student download the repository from [Student Activity 2]()*
For creating sockets, Python has a very famous and widely used library called ***socket*** As a result, we have to ***import socket*** into server.py. For the server and client to communicate simultaneously, we will be using threading to execute these functions parallely.Thus, we'll import the ***Thread*** library too.	
 ```import socket``` ```from threading import Thread```	
We've also declared some global variables such as SERVER,PORT, and IP_ADDRESS and set their values to None. None defines a null value, or no value at all. **None** is not the same as 0, False, or an empty string We are using **127.0.0.1** as IP address and 8080 as the port. however this port can be any number. Just make sure that it is not anything lower than ***1,024***, since those are reserved ports.	
 ```IP_ADDRESS = '127.0.0.1'``` ```PORT = 8080``` ```SERVER = None``` ```clients = {}```	

Next, we'll declare an empty ***clients*** dictionary that contains the information about clients that have connected to the server. As of now, it looks like this:	


```
import socket
from  threading import Thread
IP_ADDRESS = '127.0.0.1'
PORT = 8080
SERVER = None
clients = {}
```

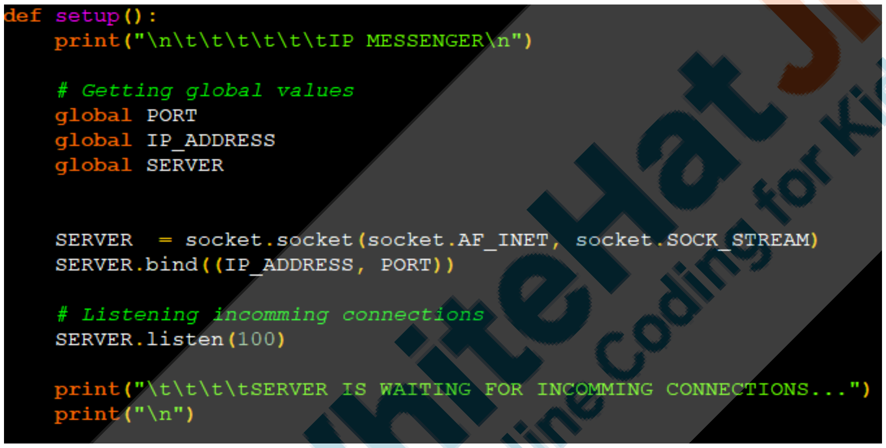
The **server.py** file now contains all the necessary information. We just need to create the server.	
A function called **setup()** was created to setup the server First, we print the name of our application IP Messenger inside the function Then we call all the global variables that we declared earlier, namely **PORT, IP_ADDRESS, and SERVER.** Then we define the IP Address and the Port we are using. Next we bind our server with the IP Address and the Port that we are using and then we are ready to listen for any incoming requests from the clients. As for the server, we're using ***socket.socket()*** function, and we're binding to it via bind(), which takes a tuple containing the ip_address and port. Our server has now successfully bound, so we can start	

listening on this server socket using the ***listen()*** function.

Here, we specify that we only want 100 connections. .

Then we are printing a text which says that the “SERVER IS WAITING FOR INCOMING CONNECTIONS.”

Then finally we’ll call the ***setup()*** wherever we need it.



```
def setup():
    print("\n\t\t\t\t\t\tIP MESSENGER\n")

    # Getting global values
    global PORT
    global IP_ADDRESS
    global SERVER


    SERVER  = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    SERVER.bind((IP_ADDRESS, PORT))

    # Listening incomming connections
    SERVER.listen(100)

    print("\t\t\t\tSERVER IS WAITING FOR INCOMMING CONNECTIONS...")
    print("\n")
```

Now the server is ready but we have to accept all incoming requests from the client side

For that we will make a function ***acceptConnections()*** and will call the same in our server ***setup()*** function

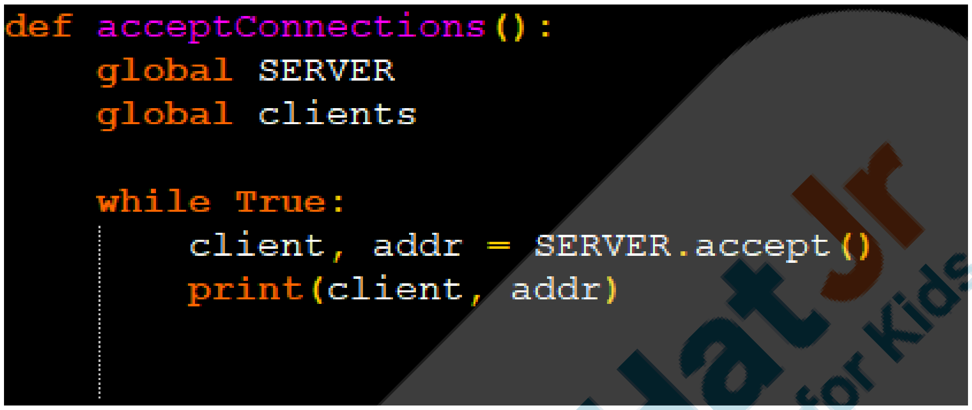
Declared global variables **SERVER & Clients**.

While the server is listening to accept the conceptions

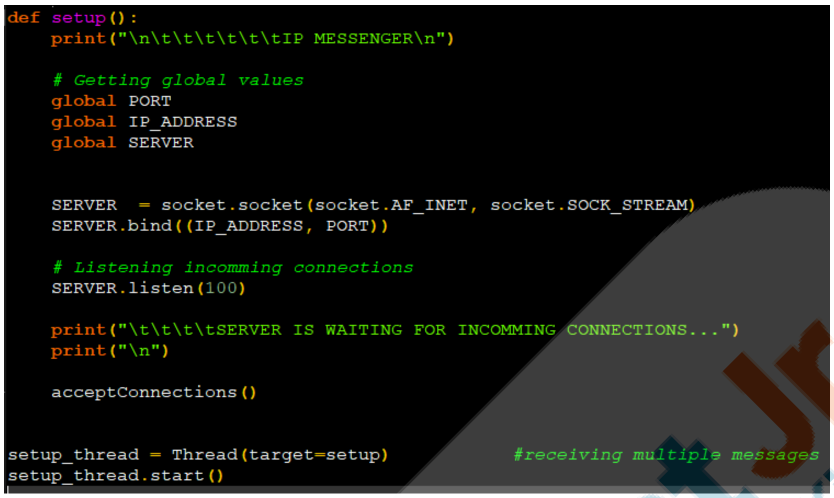
Since this indefinite process will make the while loop true, all incoming connections should be accepted using the accept() method, which waits for an incoming connection

When a client connects, it returns a new socket object representing the connection and a tuple holding the

address of the client. Now we will printing client and address	


`def acceptConnections():`
`    global SERVER`
`    global clients`
 	
`    while True:`
`        client, addr = SERVER.accept()`
`        print(client, addr)`	

It's time to call the ***acceptConnection()*** function in the **setup()** function. We don't want only a single client to connect to the server at a particular time but many clients simultaneously. We want our architecture to support multiple clients at the same time. For this reason, we must 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 ***setup()*** function to accept all the requests simultaneously and use ***start()*** to start this thread process.	

```
def setup():
    print("\n\t\t\t\t\t\tIP MESSENGER\n")

    # Getting global values
    global PORT
    global IP_ADDRESS
    global SERVER


    SERVER  = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    SERVER.bind((IP_ADDRESS, PORT))

    # Listening incomming connections
    SERVER.listen(100)

    print("\t\t\t\tSERVER IS WAITING FOR INCOMMING CONNECTIONS...")
    print("\n")

    acceptConnections()


setup_thread = Thread(target=setup)                 #receiving multiple messages
setup_thread.start()
```

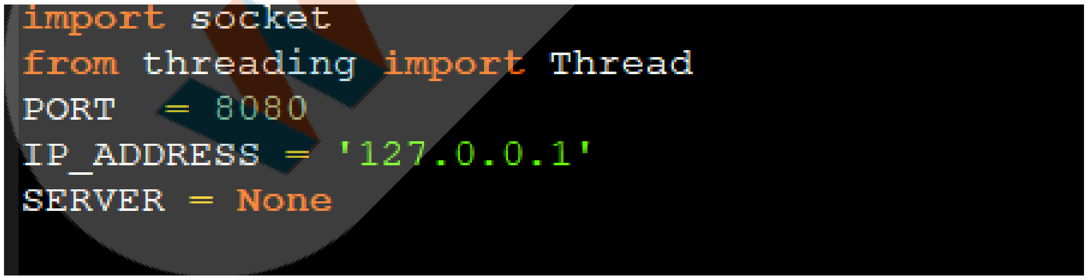
Now the server is ready to accept the connections from the client side, but we are not ready with the client part. So we won't be able to connect the server with the client.

So We'll be creating a **client.py** file .

Import socket and thread library

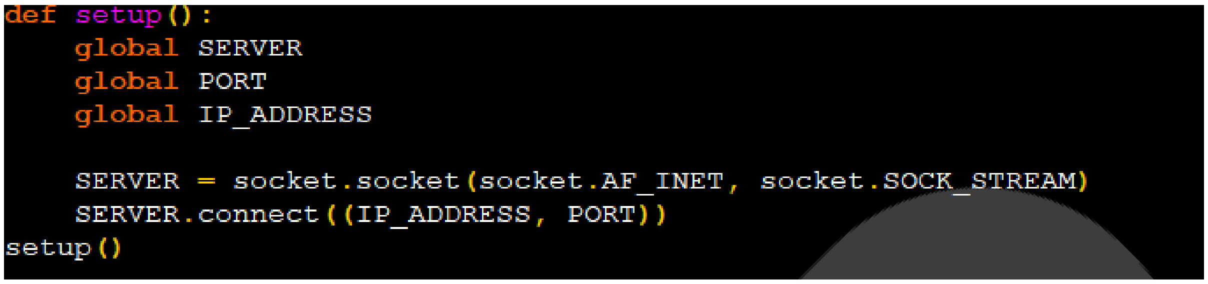
Set the same ***IP address & Port*** which we used for the server.

Declared variables **SERVER** set their value to **None**.



```
import socket
from threading import Thread
PORT    = 8080
IP_ADDRESS = '127.0.0.1'
SERVER = None
```

After that, we'll write a function ***setup()*** that sets up a socket and connects the client to the server using IP address and port.



```
def setup():
    global SERVER
    global PORT
    global IP_ADDRESS

    SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    SERVER.connect((IP_ADDRESS, PORT))
setup()
```

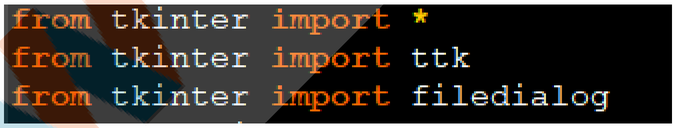
This will be the same as setting up the server earlier.
We'll write the ***setup()*** function for the client side.

- Within the function, we'll use the global variables SERVER, PORT, and IP_ADDRESS.
- Create a client-side socket.
- Using the ***connect()*** function, we establish a connection between the server and client

The boiler code ends here

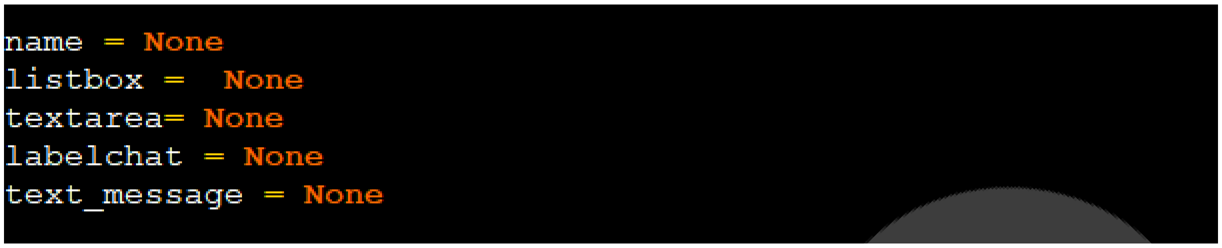
Now that the client and server files are connected, it's time to create the GUI for FTP Application.

As we know for GUI we need to import the Tkinter library on top of the code, along with Tkinter we will import other Tkinter libraries like the file dialog, which lets you open a file, a directory, save as file, and more.



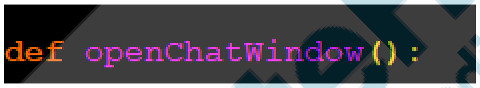
```
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
```

We will declare all variables necessary for our user interface, name listbox, textarea, labelchat, text_message and will set their value to None.



```
name = None
listbox =  None
textarea= None
labelchat = None
text_message = None
```

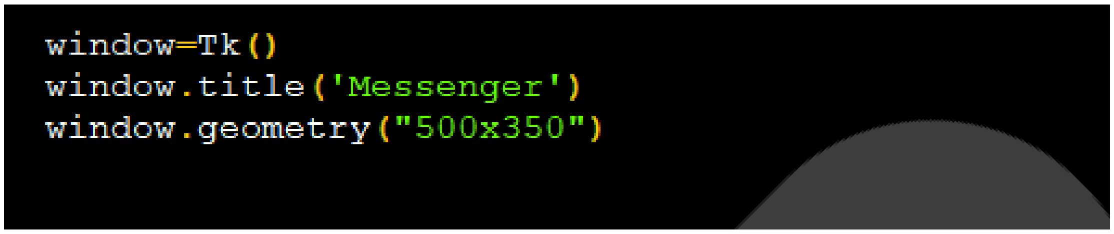
Our GUI will be displayed whenever our client side script is executed. We will create this GUI within a function. Using the function name ***openChatWindow(),*** let's begin code the for GUI	



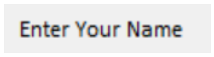
```
def openChatWindow():
```

Here is the user interface of the FTP system What kind of widgets do we need to make our GUI? Right! In order to make an FTP app , we need the following: Entry box, Button, Listbox, Listbox Scrollbar, Connect, Disconnect, test-area, text-area, scroller for test-area, Send button for message, Attach & send button.	**ESR** Buttons, Entrybox, Connect, Disconnect Refresh
Next, you initialize the window with the ***Tk()*** method and assign it to a variable. A blank window is created with close, maximize, and minimize buttons on top, as a normal GUI should.. Set the window title using ***title()*** . Let's name it "Messenger". Set the geometrical configuration of the window using	

geometry(), which in our case is ("500 * 300").

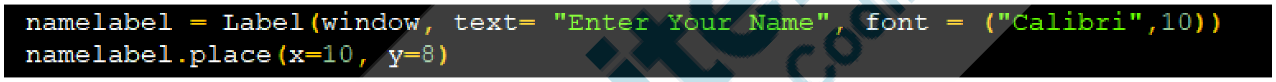


```
window=Tk()
window.title('Messenger')
window.geometry("500x350")
```

Create Name label: 

Create a name label using the **widget *Label()*.**

- Call the main window first, then set the Label attributes, such as text and font, and then place it on the screen using the ***place()*** method.



```
namelabel = Label(window, text= "Enter Your Name", font = ("Calibri",10))
namelabel.place(x=10, y=8)
```

Create Text Input:

Create an input box using ***Entry()*** widget.

- Set the attributes for Entry, call the main window first and then set attributes for the Entry i.e. width, font and then place it on the screen using ***place()*** method.

- Add focus using ***focus()*** to highlight the entry box.

```
name = Entry(window,width =30,font = ("Calibri",10))
name.place(x=120,y=8)
name.focus()
```

Create Connect to Chat Server Button:

Button() widget can be used to create a button.

- Set the Button attributes, call the main window and then set the Button attributes, such as the text and font, and then place it on the screen using ***place().*** As we move forward, we will use Command to handle clicks

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

Create Separator:

Separator() widget to create a Separator.

- Assign attributes to Separator. Call the main window first and then assign attributes to Separator, i.e. orient, and then place it on the screen by using ***place()*** method. The click event will be handled using Command.

```
separator = ttk.Separator(window, orient='horizontal')
separator.place(x=0, y=35, relwidth=1, height=0.1)
```

Connect Button :

Create a button by using the ***Button()*** widget.

- Setting the button's attributes involves calling the main window first, setting the button's attributes such as text and font, and placing it on screen using the ***place()*** method. Later, we will use Command for this click event

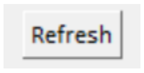
```
connectButton=Button(window,text="Connect",bd=1, font = ("Calibri",10))
connectButton.place(x=282,y=160)
```

Disconnect Button:

Using the ***Button()*** widget, create a disconnect button.

- Call the main window first, and then set the attributes for Button - i.e. text, font, and then place it on the screen using the ***place()*** method. This event will be handled by Command later on.

```
disconnectButton=Button(window,text="Disconnect",bd=1, font = ("Calibri",10))
disconnectButton.place(x=350,y=160)
```

Refresh Button 

Create a Refresh button using the ***Button()*** widget. ● Set the attributes for the button, first call the main window, and then set the text, font, and then place it on the screen using the ***place()*** method. Eventually, we'll use Command for the click event.	

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

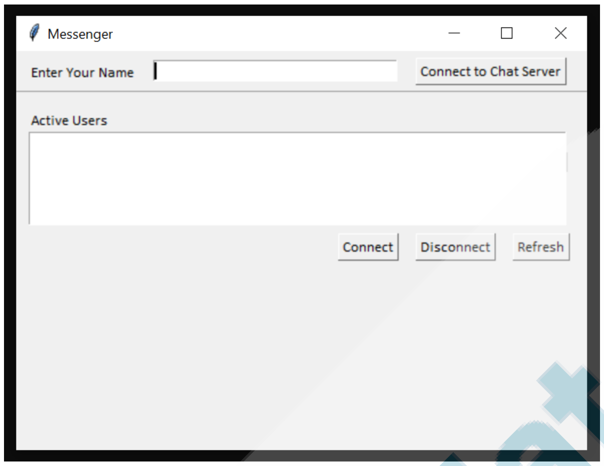
In order to make your U-I visible on screen, we must call mainloop.	

```
window.mainloop()
```

The first time we made GUI, we created an open chat function and put all the design widgets() inside it. Now, let's call this function in the client ***set up()*** function, so that whenever we run a client script, this display will be visible	

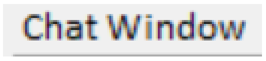
```
def setup():
    global SERVER
    global PORT
    global IP_ADDRESS

    SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    SERVER.connect((IP_ADDRESS, PORT))


    openChatWindow()

setup()
```

So far, our output looks like this:



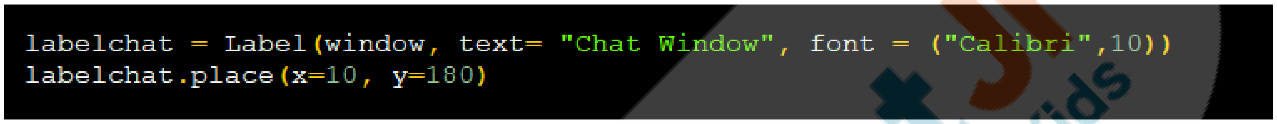
It's now your turn to design the rest of the window.

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.**	
ACTIVITY • **Create Chat label** • **Adding Text area, scroll bar** • **Create Input Box for message** • **Create send & attach button**	
Teacher Action	**Student Action**
Guide the student to get the boilerplate code from Student Activity 2	*Student clones the code from Student Activity 2*

Chat Label 
Chat Window

Using the ***Label()*** widget, create a chat name label.

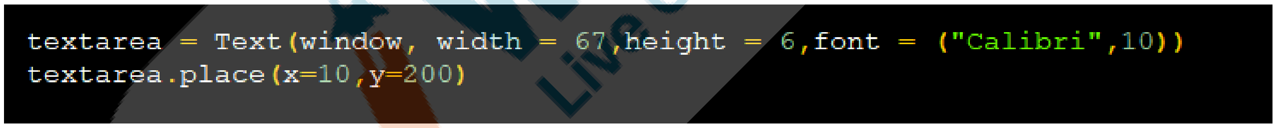
- Call the main window first and then set the Label's attributes, such as text and font, and then place it on the screen using the ***place()*** method.



```
labelchat = Label(window, text= "Chat Window", font = ("Calibri",10))
labelchat.place(x=10, y=180)
```

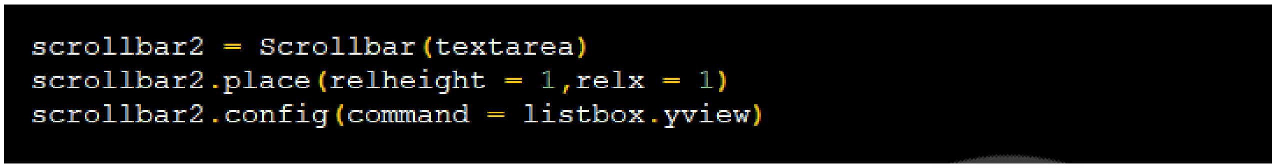
Use the ***Text ()*** widget to create a Text Area.

- Setting the attributes for a Scroll Bar, calling the main window, setting the attributes for a Text, such as the size, height, and font, and then placing the text on the screen with the ***place()*** method.



```
textarea = Text(window, width = 67,height = 6,font = ("Calibri",10))
textarea.place(x=10,y=200)
```

Vow, we have done so much today Great! So can you tell what other widget we need to show on user-interface *Teacher helps the student in writing the code*	**ESR** *Entry box, buttons* *Student writes the code*

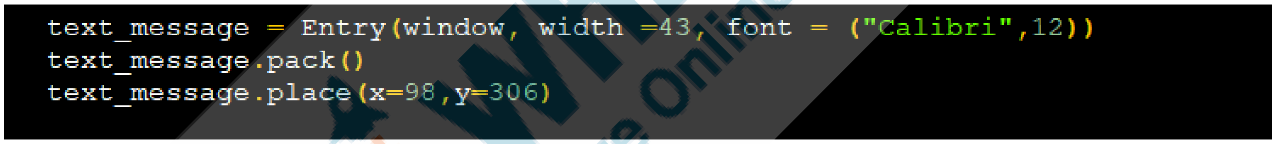


```
scrollbar2 = Scrollbar(textarea)
scrollbar2.place(relheight = 1,relx = 1)
scrollbar2.config(command = listbox.yview)
```

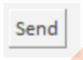
Message input to write messages

Use the ***Entry()*** widget to create a message input box.

- set attributes for the Entry by calling the main window first, then setting width and font attributes, and then putting it on the screen by using the ***place()*** method. Use the ***pack()*** method to pack the box.

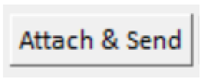


```
text_message = Entry(window, width =43, font = ("Calibri",12))
text_message.pack()
text_message.place(x=98,y=306)
```

Send Button: 

Create a send button using the ***Button()*** widget.

- Call the main window first, and then set the attributes for Button - i.e. text, font, and then place it on the screen using the ***place()*** method. We will later use Command for this click event.

Attach & Send Button: 

To fetch files from the computer, we need an attach & send button. Create an attach button using the ***Button()*** widget. Call the main window first, and then set attributes for the button, such as text and font, before placing it on the screen using ***place()*** method. We will later use Command for this click event.

```
attach=Button(window,text="Attach & Send",bd=1, font = ("Calibri",10))
attach.place(x=10,y=305)
```

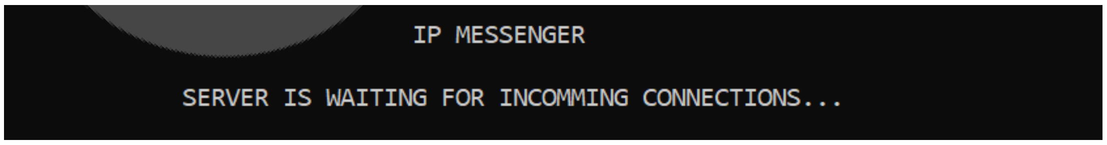
File path label

This label will display the path of the fetched file.

- Create a file path label using ***Label()*** widget. Set the attributes for Label, call the main window first and then set attributes for the Label i.e. text, font and then place it on the screen using ***place()*** method.

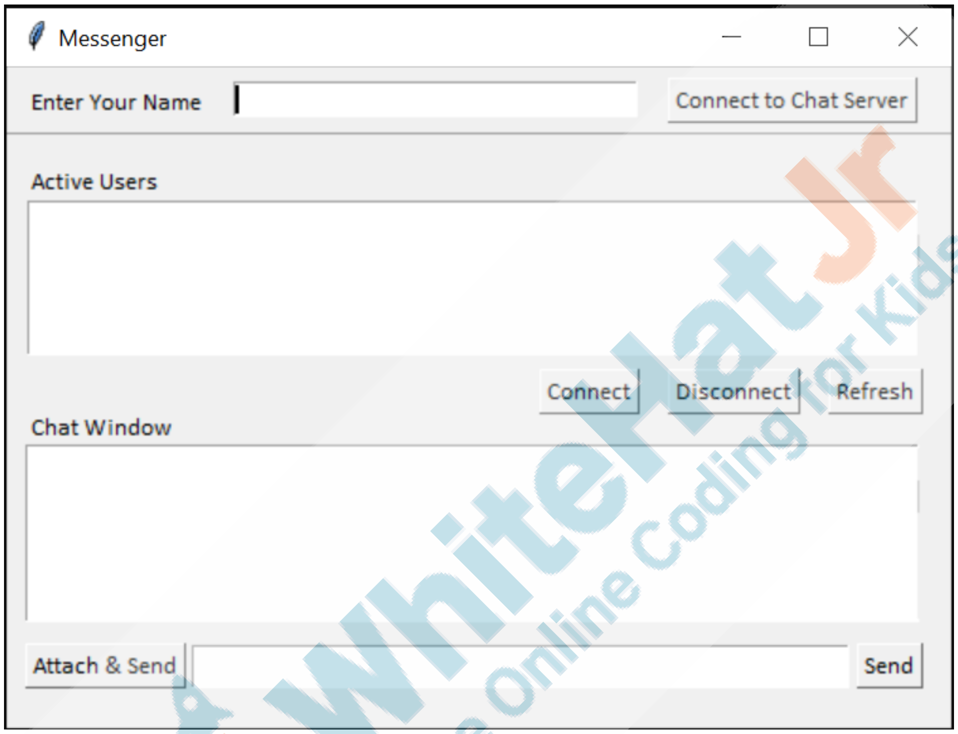
```
filePathLabel = Label(window, text= "",fg= "blue", font = ("Calibri",8))
filePathLabel.place(x=10, y=330)
```

So we are done with the design part of our interface. It's time to see output ***Teacher helps the student in running the files***	***ESR*** ***Student runs the files***
server.py in terminal/cmd looks like -	



```
                 IP MESSENGER

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

client.py in the terminal/cmd looks like -

IP MESSENGER	
A window has appeared as well, that appears to be - 	
It's amazing! We have completed the first part of the app! We will now work on the button's functionality in the next class	
Teacher Guides Student to Stop Screen Share	
WRAP UP SESSION - 5 Mins	
Quiz time - Click on in-class quiz	
Question	**Answer**
File Transfer Protocol is built on which architecture ? A. Client Only	**B**

B. Server-Client C. Server Only D. Data-base	
To accept connection from the client what two important parameters are required at the server side? A. Host,port B. Addr C. Host D. part	**A**
Which widget is used to create separation on the interface screen? A. Separate() B. Separation() C. Difference() D. Separator()	**D**

End the quiz panel

FEEDBACK

- **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	*Make sure you have given at least 2 Hats Off during the class for:*

WhiteHat Jr
Live online Coding for Kids

	Strong Concentration +10
Project Discussion We created a File Sharing application part one. In class we made a socket -client connection and created a GUI for file sharing application. In this project, you will again have to make a socket -client connection first and then we will develop a music sharing app. **Story:** Maria enjoys listening to music. With online apps, she gets bored quickly. Her goal is to create her own music desktop application, so whenever she becomes bored, she can listen to a song, download a playlist, or even create a new playlist. Your task is to create a server and client connection and create a graphical interface for a music sharing app.	
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 student uses the markdown editor to write her/his reflections in the reflection journal.*

○ 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?	

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	IP Messenger	https://en.wikipedia.org/wiki/Windows_Messenger_service
Teacher Activity 2	Boilerplate Code	https://github.com/pro-whitehatjr/PRO208_TeacherActivity_Boilerplate
Teacher Activity 3	Reference Code	https://github.com/pro-whitehatjr/PRO-208_ReferenceCode
Student Activity 1	IP Messenger	https://en.wikipedia.org/wiki/Windows_Messenger_service
Student Activity 2	Boilerplate Code	https://github.com/pro-whitehatjr/PRO208_StudentActivity_Boilerplate
Teacher Reference In-Class-Quiz	In-Class-Quiz	https://s3-whjr-curriculum-uploads.whjr.online/375f8e8

		8-2c83-43fc-b7a1-e4167a79b621.pdf