Topic	GUI BASED CHAT APP - 2	
Class Description	**Students will add a GUI to the client app they built and deep dive into Tkinter library**	
Class	**C-203**	
Class time	**45 mins**	
Goal	● Understand about Client Programming ● Building Client in Python ● Application on Socket	
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

WhiteHat Jr
Live online Coding for Kids

Question	Answer
What does Thread mean in Programming? A.Separate flow of execution B.Continue flow of execution C. Single flow of execution D. No flow of execution	**A**
Which below option will test a block of code for errors? A. try B. thread C. except D. None of the above	**A**

TEACHER-LED ACTIVITY - 10mins

Teacher Initiates Screen Share

ACTIVITY

- **Creating Chat App Screen and their functions**

Teacher Action	Student Action
In the last class, we created login GUI for chat app Any doubts from the last session? *Teacher resolves the query of the student (if any)* Great! Can you tell me how we did in the last class?	**ESR:** Varied! **ESR:** We did it using the Tkinter library. We create a top level window for our login screen
Great!	

Now in the last session we created User -Interface for the login part only? But that one is not complete as that interface only allows us to do login for the Chat interface. But where can we connect with each other? For that we need to design another screen for chat Can you tell me how we can design our chat window? Perfect! So, can we start making another user-interface for the chat part?	**ESR** **Tkinter!**
In the last class we created Login window and we discussed "top Level" window and Main window " We created a Login window i.e., top Level window now it's time to create the main screen. Why class? You must be thinking why we need a function for creating a chat window? Right! As we know this chat app is multi - client chat app, we need a separate chat window for all clients. So, whenever we need, we can use our class for making User-Interface for another client for creating our class objects.	
Let's get into our ***client.py*** and start with the Layout *Teacher clones the repository from [Teacher Activity 1]* This is the same code from C-202 and chat window layout have been added	*Student clones the repository from [Student Activity 1]*

Let's say that we create a function called ***layout***, which will be for our chat window. This will be able to handle the receiving and writing of the messages too!

Here this function will need two arguments- ***self*** and ***name***.

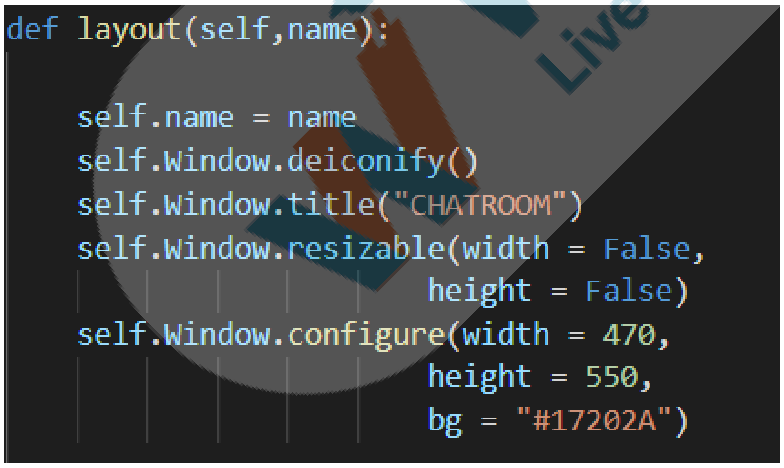
As we already know, ***self*** will be referred to as an object for the chat window. So, while creating chat -window, will use self with all widgets

To display the chat window after the logic window we are using the **deiconify()** method.

deiconify() method will raise this chat window and give it the focus after top level window

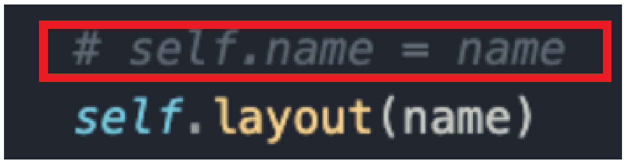
Set title for window using **title(),** resize it using **resizable()**, it configures the frame window size resizable and helps to freeze window using width & height equal to False

Configure the window with height, width and background color.



```
def layout(self,name):

    self.name = name
    self.Window.deiconify()
    self.Window.title("CHATROOM")
    self.Window.resizable(width = False,
                          height = False)
    self.Window.configure(width = 470,
                          height = 550,
                          bg = "#17202A")
```

Also, since we are keeping our ***self.name = name*** line here, let's comment it from our ***goAhead()*** function, and call the ***self.layout()*** function there -



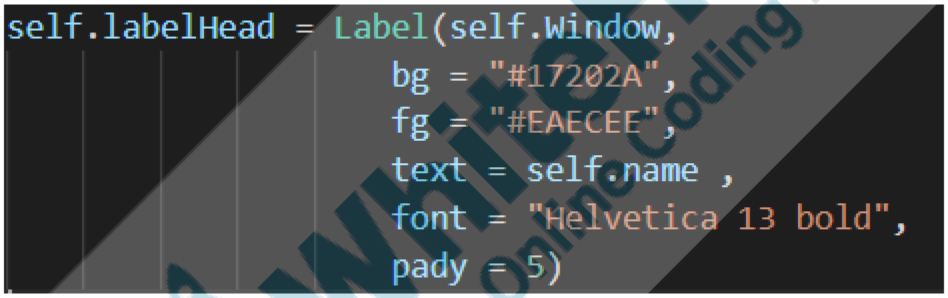
```
# self.name = name
self.layout(name)
```

Okay, now our main chat window is going to have a name at the top.

Let’s work on the name label

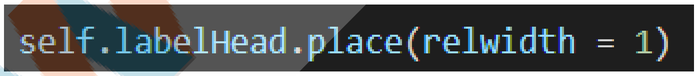
Can you help me out with the code?

Add a **Label()** widget for showing the name of the client. This label is for the main chat window i.e., we are using self. Window, set background color, foreground color ***#EAECEE***. Here we are using HTML color codes instead of color names.Use # with color code whenever we use HTML color codes.Set the font and padding from the top i.e Y-axis.



```
self.labelHead = Label(self.Window,
                       bg = "#17202A",
                       fg = "#EAECEE",
                       text = self.name ,
                       font = "Helvetica 13 bold",
                       pady = 5)
```

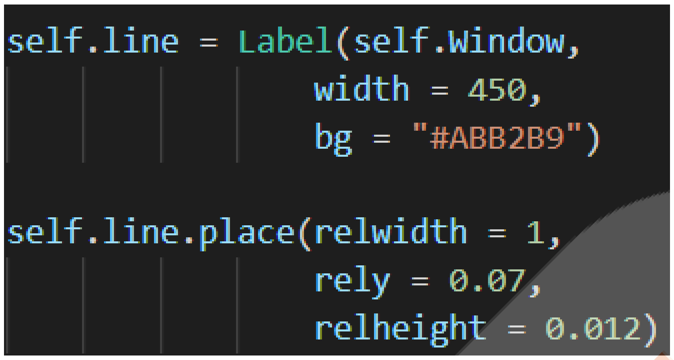
Here, it's time to place your label Head, you can see that we have used ***relwidth*** in the ***place()*** function. ***relwidth*** means the relative width of the widget, ***Label*** in our case.



```
self.labelHead.place(relwidth = 1)
```

It’s our chat interface so why not to make it more attractive. Let's give some effects.

Create one line after the user name so it can create little difference in visual effects.Its very easy just you need to create one Label() widget for window , set width and color and then we need to place it using **place()** function,

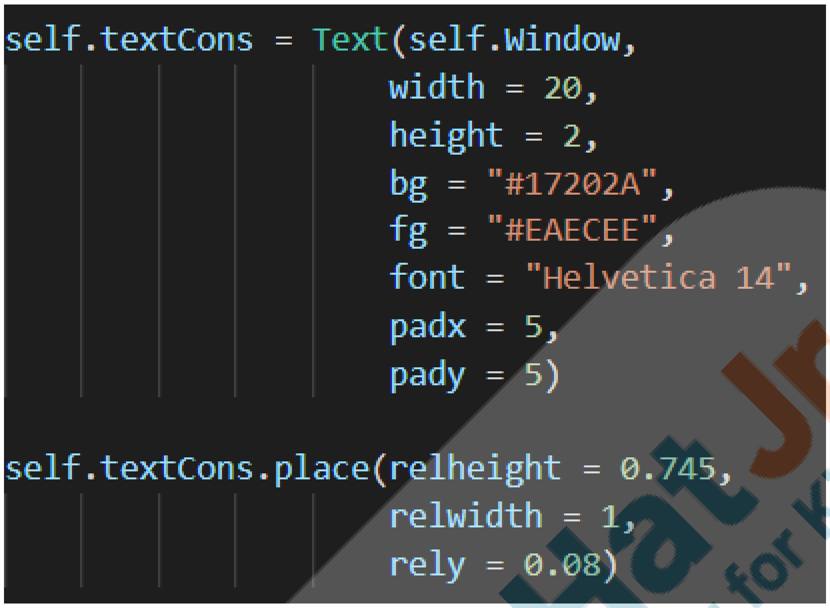


```
self.line = Label(self.Window,
                  width = 450,
                  bg = "#ABB2B9")

self.line.place(relwidth = 1,
                rely = 0.07,
                relheight = 0.012)
```

Okay, now our chat window has user name Now let's think what else do we need? Can you help me in writing the code? *Teacher writes the code with the student's help*	**ESR:** We need a chatting text area,text input.We also need a button to send the message *Student helps the teacher in writing the code*

We need to create a text area where all clients can chat. For that we will use a text widget. **Text()** Widget is used where a user wants to insert multiline text fields as messaging, sending information or displaying information and many other tasks.
Set the width, height, along with background,foreground color, font and font size, and give padding from both sides i.e. x and y axis.

After creating text widget, place it on the chat screen using **place()**



```
self.textCons = Text(self.Window,
                     width = 20,
                     height = 2,
                     bg = "#17202A",
                     fg = "#EAECEE",
                     font = "Helvetica 14",
                     padx = 5,
                     pady = 5)

self.textCons.place(relheight = 0.745,
                    relwidth = 1,
                    rely = 0.08)
```

We've achieved quite a lot! Now all we need a bottom panel where client can type his or her message and along with that we need one button Button will behave like an event. On clicking the button, enable the user to send the message, and also, we will need a function that is called when the button is clicked to handle the event and one function for sending the message and one function for writing the message. I think we need a scroll bar also to move our main text area. Let's design all It's your turn from here.	

Teacher Stops Screen Share

STUDENT-LED ACTIVITY - 20 mins

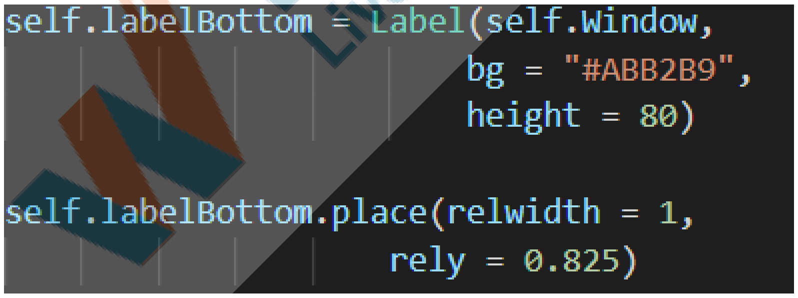
- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Fullscreen.**

ACTIVITY

- **Create Bottom panel for sending message**
- **Add scroll bar to move chat text area**
- **Add functions for sending message**

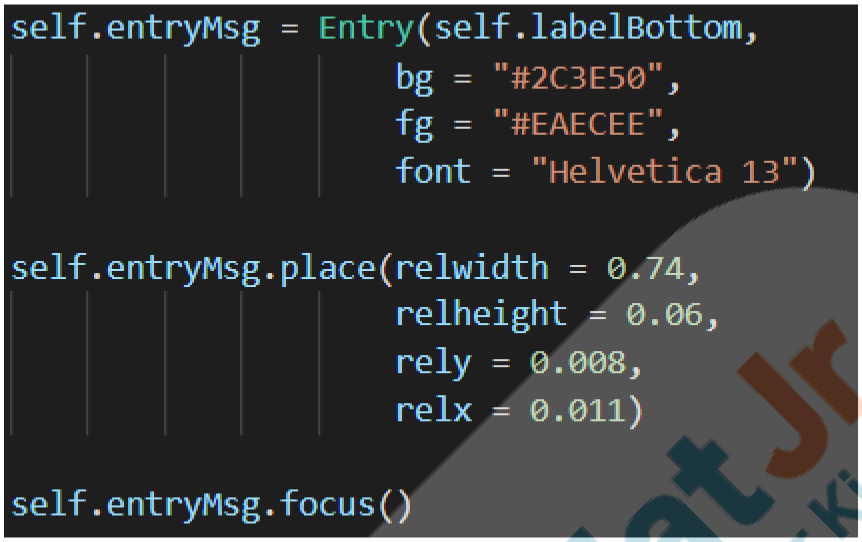
Teacher Action	Student Action
Guide the student to get the boilerplate code from Student Activity 2	*Student clones the code from Student Activity 2*
Alright, let's start by adding the bottom panel for sending a message. For bottom panel we need one label where we will use one input box and one button for sending message *Teacher helps the student in writing the code*	**ESR:** *Student writes the code*

Let's start with labels first. Create one label widget using Label() and then set it on the chat window using place() method. This Label is for the main window as we know, set background, and height for the label and then place label at appropriate location using place() x,y coordinates.This time we are using little more height as we need to insert out Entry() widget for taking input from the user and one button() widget for sending message.



```
self.labelBottom = Label(self.Window,
                         bg = "#ABB2B9",
                         height = 80)

self.labelBottom.place(relwidth = 1,
                       rely = 0.825)
```

Alright, now add one text inside your label area. Remember this time we are creating ***Entry()*** Widget inside **label()** means, it should be displayed inside label bottom, instead of window, we will write label bottom.Set argument for your ***Entry()*** widget like background color,foreground color and font along with size.Place it on appropriate location. Set relwidth, relheight ie. relative width and height, relative y and relative x position.



```
self.entryMsg = Entry(self.labelBottom,
                      bg = "#2C3E50",
                      fg = "#EAECEE",
                      font = "Helvetica 13")

self.entryMsg.place(relwidth = 0.74,
                    relheight = 0.06,
                    rely = 0.008,
                    relx = 0.011)

self.entryMsg.focus()
```

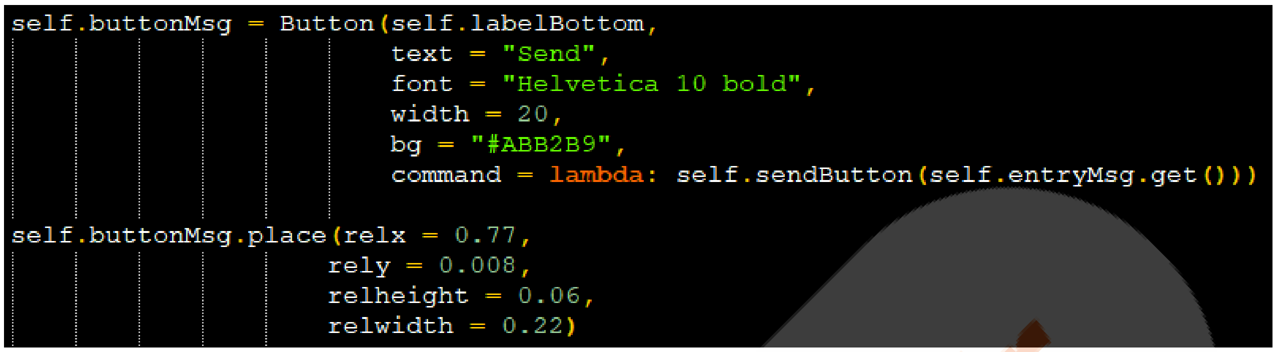
Look, we need to focus our entry() widget for that we will use **focus()**. focus() will display the cursor at your entry() widget.

Add a send button, use button widget()
Here, a button object is created with the ***Button()*** class and is saved in a variable called button msg.

Now this ***Button()*** class took the first argument, the labelbottom in which it should be displayed, then it took the ***text***, ***font*** it should use, width of the button, background color of the button then finally, it took an argument called a ***command***.

For the ***command*** argument, it is the command the button should follow in case it is clicked.

We can use a special keyword called ***lambda***, which is a one line function definition, just like arrow functions in JavaScript. Having ***self.SendButton()*** as a function itself will stop it from getting executed until the button is clicked



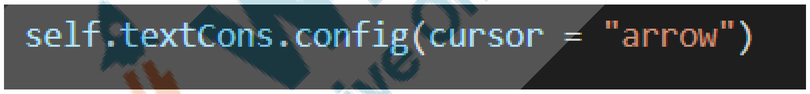
```
self.buttonMsg = Button(self.labelBottom,
                        text = "Send",
                        font = "Helvetica 10 bold",
                        width = 20,
                        bg = "#ABB2B9",
                        command = lambda: self.sendButton(self.entryMsg.get()))

self.buttonMsg.place(relx = 0.77,
                     rely = 0.008,
                     relheight = 0.06,
                     relwidth = 0.22)
```

Now this can also be like the button's click handler. We want to call a function here that can handle the click situation.

This function is required to be a part of the chat layout, so we are using the keyword ***self*** with function.Now, on calling this function will take arguments, one will be the self and other be the message the user entered. We can get it from the entry widget we created just now with the ***get()*** function.

This means that the ***sendButton*** function will be called as soon as you click the button.

We need to show our cursor in the ***Entry()*** widget as we already use focus() along with that will use the ***cursor*** attribute too.



```
self.textCons.config(cursor = "arrow")
```

Now, let's do a little brainstorming!

What if the chat text area is filled with all messages, then what can we do ? How can we see all messages in one go?

Do you have any ideas for this ?

Any ideas?

Whenever you write code sometime you want to see up in the screen and sometime down

Right !

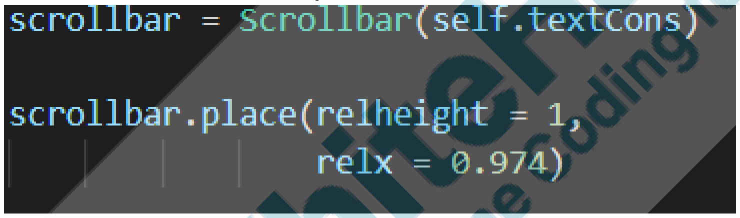
So that time we use scroll bars ?

Exactly, here in this chat screen there is a need for one scroll bar in the text area.

Let’s make one scroll bar !

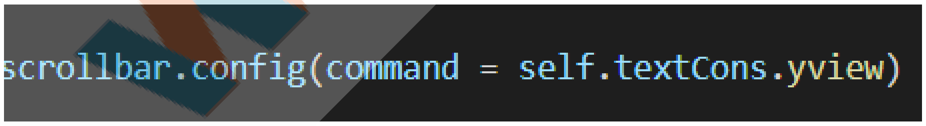
The scrollbar widget is used to scroll down the content. We can also create the horizontal & vertical scrollbars to the text widget.

Use scrollbar widget save that widget in one variable, Look we need to display scroll bar in chat textarea vertically ie so we will pass the argument textCons inside the scroll bar. self here is referred to as this chat window. Place the scrollbar(). Set the relative height, for height of the scroll bar and relative x position.



```
scrollbar = Scrollbar(self.textCons)

scrollbar.place(relheight = 1,
                relx = 0.974)
```

Let’s put some light on it when your scroll bar will be active ?
Only when you want to see messages up and down on screen, so this will act as an event handler so we are using commands for the same.
Config the scroll bar using ***config()* and** active with command which will be equal to textcons because it will be active only whenever the text is present in the textarea and we want to see vertically so we are using yview



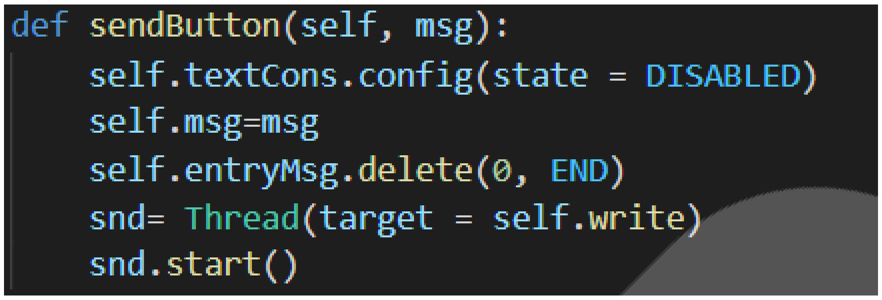
```
scrollbar.config(command = self.textCons.yview)
```

Now we are done with our design parts let’s move forward how things will work behind the backend

Think about how buttons work on click ?

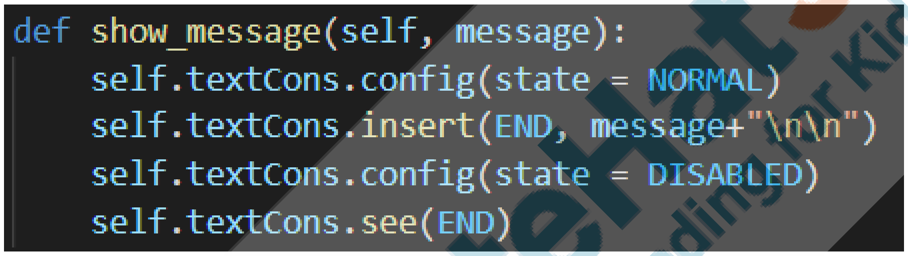
How messages will display in the chat area?

How can we write the text in the input box i,e Entry widget ? Now let's start writing the functions- *Teacher helps the student in writing the code*	*Student writes the code*
Let's create function for button Here, we have defined a function called ***send button*** inside the class with 2 arguments - 1. ***self*** - Because this function belongs to the window itself 2. ***msg*** - Contains the message that the client will send Inside this function, the first thing that we do is need to disable the text widget. In **Tkinter**, sometimes, we may want to make a **text** widget **disabled**. To achieve this, we can set the **text** configuration as **DISABLED**. This will freeze the **text** widget and will make it read-only. Create one variable and save a client message that will get from the client. After one message we need to empty the entry () widget also. So that user can enter data again. Use delete() function to delete the data from start to end. Passing two index arguments i.e start and end point. Delete the data starting from 0 till end. As the sending message will continue, the process initiates a thread for writing a message. Create a new ***Thread()*** and call a function called ***self.write*** inside it by setting it as the ***target***. Start the thread process using ***start ()*** function.	



```
def sendButton(self, msg):
    self.textCons.config(state = DISABLED)
    self.msg=msg
    self.entryMsg.delete(0, END)
    snd= Thread(target = self.write)
    snd.start()
```

Let’s create another function
We need to show a message   inside our text widget.



```
def show_message(self, message):
    self.textCons.config(state = NORMAL)
    self.textCons.insert(END, message+"\n\n")
    self.textCons.config(state = DISABLED)
    self.textCons.see(END)
```

Let’s create another function to display a message inside our text widget.
Here, we have defined a function called ***show message*** inside the class with 2 arguments -

3. ***self*** - Because this function belongs to the window itself
4. ***message*** - Contains the message that need to show on display

Inside this function, the first thing that we do is set the text widget in the normal state as earlier it was in disable state.

In **Tkinter**, sometimes, we may want to make a **text** widget in a normal **state** . To achieve this, we can set the **text** configuration as **NORMAL**. The button could be clicked by the user this time.

Show the message inside text using ***insert().***

After getting message disable the text widget

Display the complete message . As we learnt while indexing, start can be optional but stop is compulsory.

Now we just left with one function.

Let’s create another function to write a message inside our Entry()
Here, we have defined a function called ***write*** inside the class with 1 argument -

self - Because this function belongs to the chat window

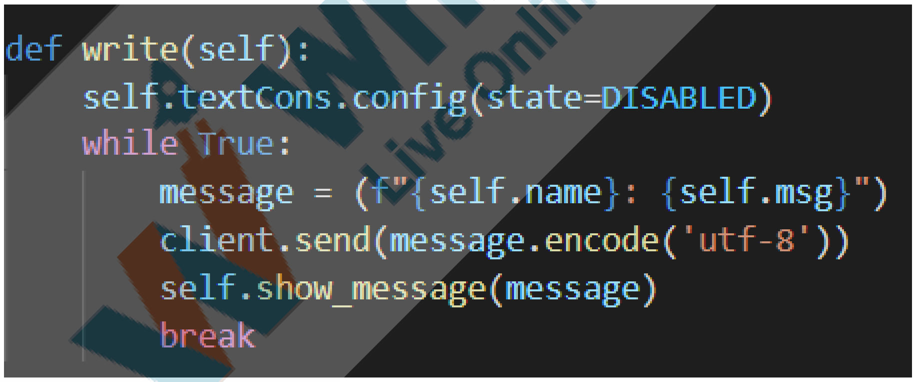
Inside this function, the first thing that we do is set the text widget in the disablel state again as the client is typing a message in the Entry box.

Disable will freeze the user text widget means it's not clickable.

Create a while loop for this. Show the client name along with the message which is stored inside the message which we have been used in show message .

Send the client message using **send()** function and encode the message in”UTF-8”format and then call the showmessage function with an argument i.e message from the client.

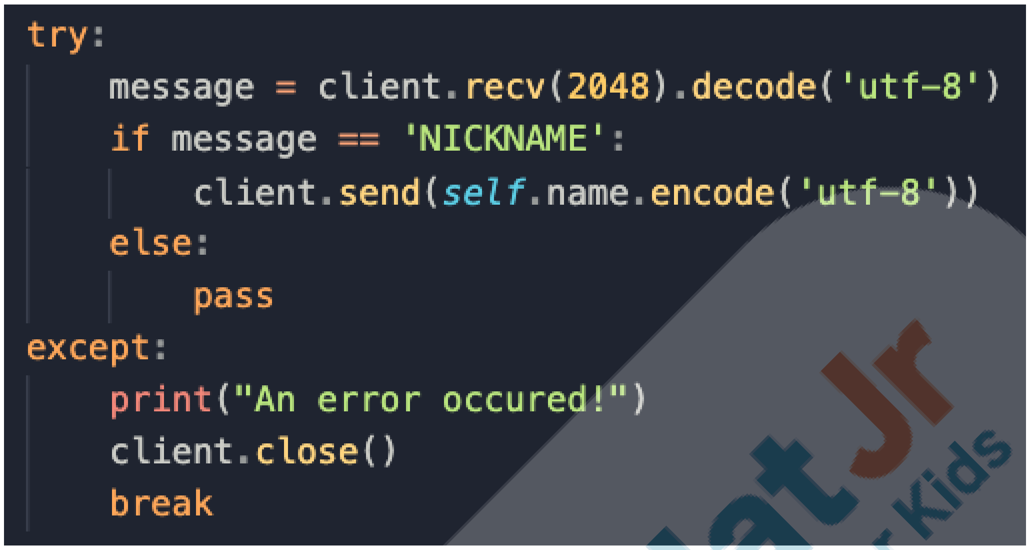
Break the loop once if the process is done.



```
def write(self):
    self.textCons.config(state=DISABLED)
    while True:
        message = (f"{self.name}: {self.msg}")
        client.send(message.encode('utf-8'))
        self.show_message(message)
        break
```

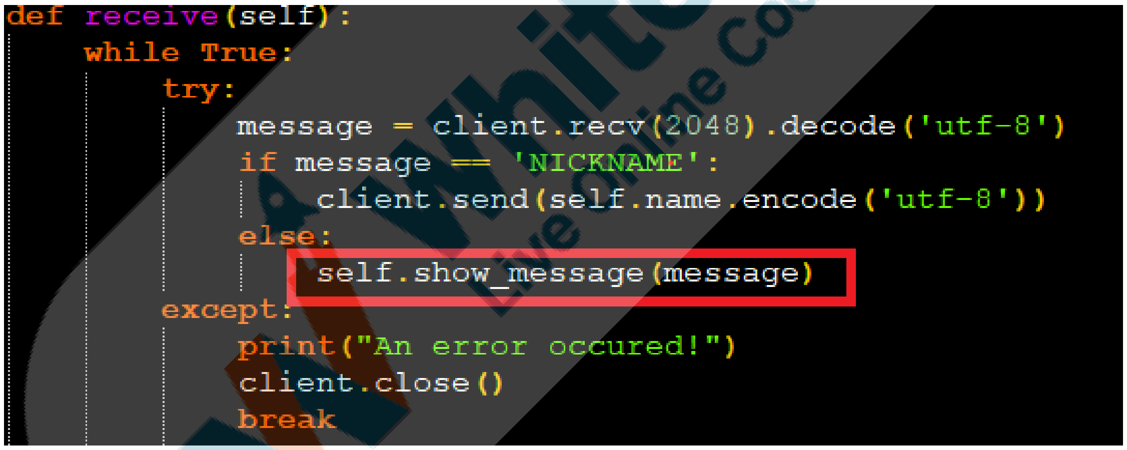
Now you remember in the last class we created receive() function. Go to receive() function.

Earlier it was like this :



```
try:
    message = client.recv(2048).decode('utf-8')
    if message == 'NICKNAME':
        client.send(self.name.encode('utf-8'))
    else:
        pass
except:
    print("An error occured!")
    client.close()
    break
```

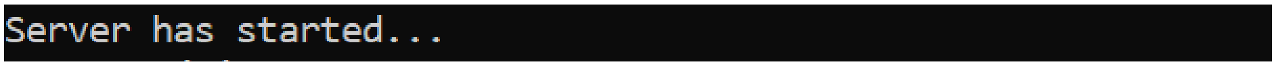
But now will just add need to else statement as we want to display our message, remove the pass and call the show_message function.



```
def receive(self):
    while True:
        try:
            message = client.recv(2048).decode('utf-8')
            if message == 'NICKNAME':
                client.send(self.name.encode('utf-8'))
            else:
                self.show_message(message)
        except:
            print("An error occured!")
            client.close()
            break
```

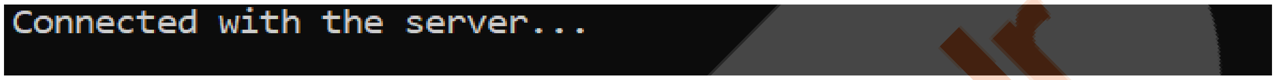
Now let’s try running the program. Run the ***server.py*** in a separate command prompt/terminal, and run the ***client.py*** in a separate command prompt/terminal ***Teacher helps the student in running the files***	***Student runs the files***

server.py in terminal/cmd looks like -



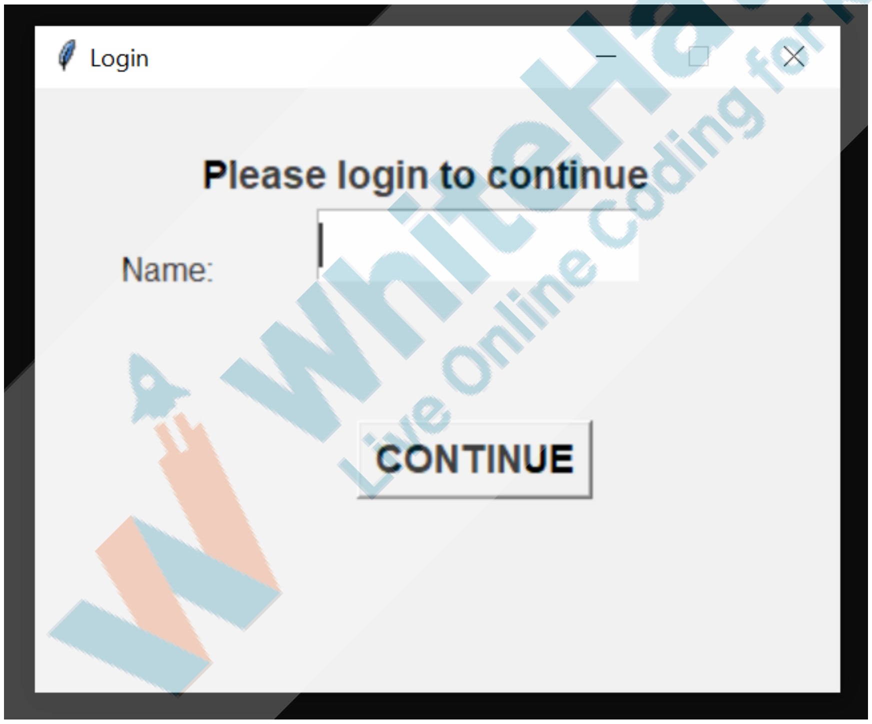
```
Server has started...
```

client.py in the terminal/cmd looks like -



```
Connected with the server...
```

And we now also have a small little window opened, that looks like -

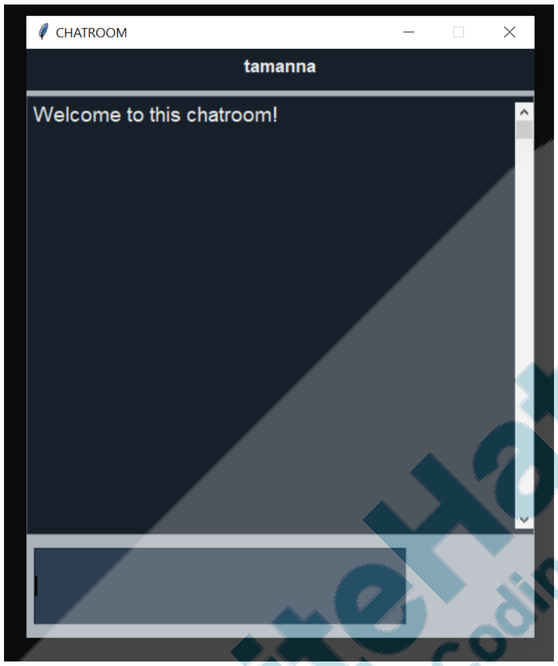


Login

Please login to continue

Name:

After enter your name it will display chat window which will look like this:



Welcome to this chatroom!

Vow! Our chat app is complete!

Isn't it fun if we can use our mobile phones too for chat apps?

No worries!

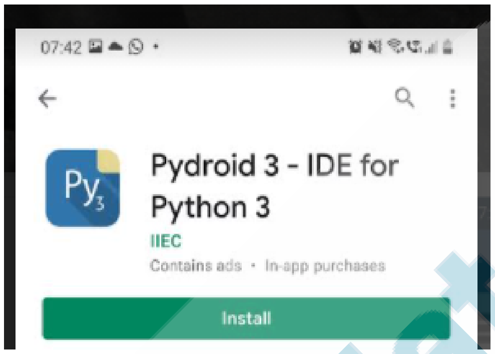
We have a solution for the same. You know coders can do anything.

Right!

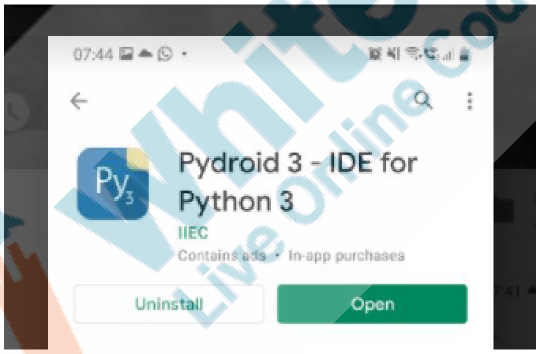
I will tell you easy steps to check this chat app on the phone too.

But before that you can send a client side file to your phone.

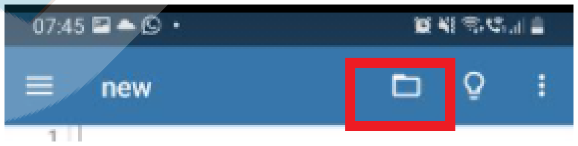
- Go to your Android app store
- Download Pydroid3



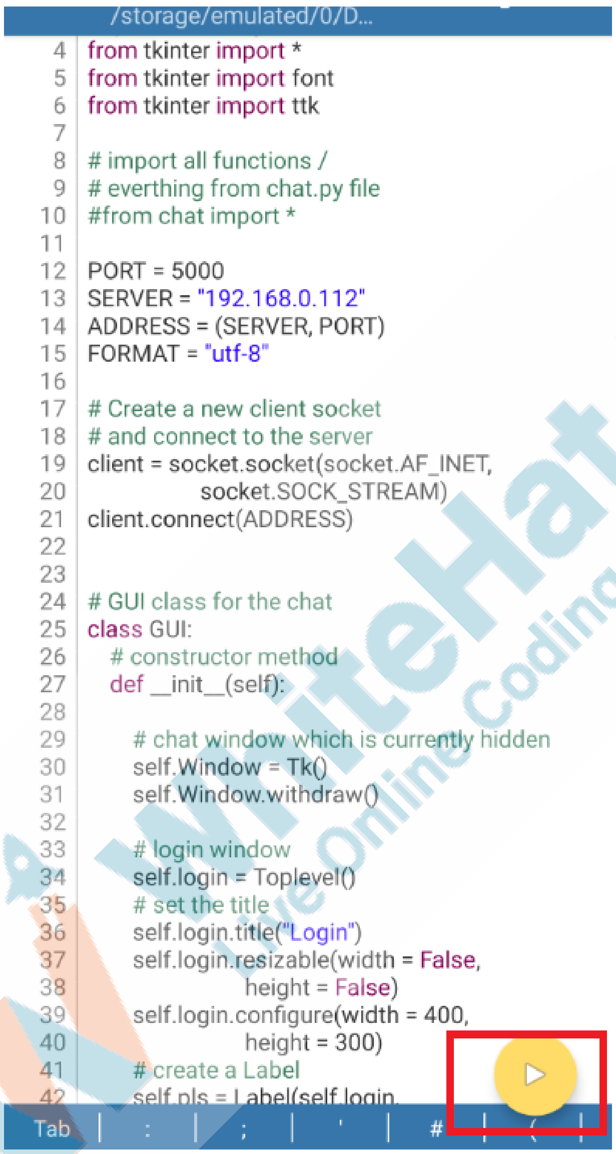
- Open Pydroid 3



- Click on rectangle icon, You will get open option



- Click on open
- Go to your phone download directory
- Upload your client side file i.e. client.py
- Click on the yellow run button at the bottom of the screen.



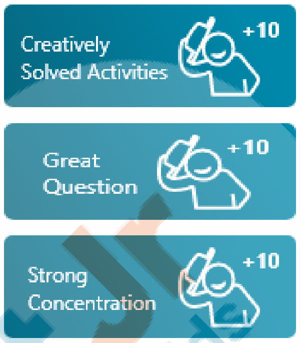
```
/storage/emulated/0/D...
from tkinter import *
from tkinter import font
from tkinter import ttk

# import all functions /
# everthing from chat.py file
#from chat import *

PORT = 5000
SERVER = "192.168.0.112"
ADDRESS = (SERVER, PORT)
FORMAT = "utf-8"

# Create a new client socket
# and connect to the server
client = socket.socket(socket.AF_INET,
                socket.SOCK_STREAM)
client.connect(ADDRESS)


# GUI class for the chat
class GUI:
   # constructor method
   def __init__(self):

      # chat window which is currently hidden
      self.Window = Tk()
      self.Window.withdraw()

      # login window
      self.login = Toplevel()
      # set the title
      self.login.title("Login")
      self.login.resizable(width = False,
                  height = False)
      self.login.configure(width = 400,
                  height = 300)
      # create a Label
      self.pls = Label(self.login,
```

Check your output on the phone too.

Teacher Guides Student to Stop Screen Share

WRAP UP SESSION - 5 Mins

Quiz time - Click on in-class quiz	
Question	**Answer**
Which one is the first step to create a parent window using Tkinter python? A. Config() B. Title() C. Call Tk() D. Geometry()	**C**
Window inside another parent window is possible by which method? A. Button() B. Container() C. Window() D. Frame()	**D**
What is the purpose of Tkinter ()? A. Develop Database B. Develop GUI C. Develop Operating System D. Develop Operating System	**B**
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:*  Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
Project Discussion	
Teacher Clicks	
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.*

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Boilerplate Code	https://github.com/pro-whitehatjr/PRO-202_Solution
Teacher Activity 2	C203 Reference Code	https://github.com/pro-whitehatjr/PRO-C203_ReferenceCode
Student Activity 1	C-202 Reference Code	https://github.com/pro-whitehatjr/PRO-202_Solution/blob/main/client.py
Student Activity 2	Boilerplate Code	https://github.com/pro-whitehatjr/PRO-C203_Boilerplate