GUI BASED CHAT APP

What is our GOAL for this CLASS?

In this class, we have applied our knowledge to add a GUI to the chat app we built earlier using Tkinter. We break the app down into two parts, the login window and the chat interface and work on the login window using Object Oriented Programming.

What did we ACHIEVE in the class TODAY?

- Added a GUI to our chat app
- Built the login window for users to enter their nickname

Which CONCEPTS/ CODING BLOCKS did we cover today?

- Tkinter
- Object Oriented Programming
- ***self*** keyword

How did we DO the activities?

In the last class, we learnt about Tkinter and how it's used to create GUI based applications for desktop. Today, we add the login window to our chat app

Activity:

1. In the code that we completed in C-200, open ***client.py*** and comment out the line for taking nickname input, the ***receive()*** and the ***write()*** function and the threads that are starting these functions -

```
import socket
from threading import Thread

# nickname = input("Choose your nickname: ")
```

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

```
# def write():
#     while True:
#         message = '{}: {}'.format(nickname, input(''))
#         client.send(message.encode('utf-8'))
```

```
# receive_thread = Thread(target=receive)
# receive_thread.start()
# write_thread = Thread(target=write)
# write_thread.start()
```

2. Import the Tkinter module -

```
import socket
from threading import Thread
from tkinter import *
```

3. Create a class called ***GUI***, and an **__init__()** function inside it to initialise the class objects. Create a window in the **__init__()** function with the **Tk()** class -

```
class GUI:
    def __init__(self):
        self.Window = Tk()
```

4. In Tkinter, we can create a main window with the ***Tk()*** class, or we can create a top level window for secondary things with the ***Toplevel()*** class. The difference between the two is in terms of the functionality and features that they offer -
    a. Since our main screen is the chat interface, we will keep the ***self.Window*** that is a ***Tk()*** object for the chat interface.
    b. Our secondary screen is the login screen, where the user enters their nickname. It will be a top level screen created with the ***Toplevel()*** class.
5. We will use the ***withdraw()*** function on the ***Tk()*** object to withdraw that window from the view. We will also create a new top level window for login screen -

```
class GUI:
    def __init__(self):
        self.Window = Tk()
        self.Window.withdraw()

        self.login = Toplevel()
        self.login.title("Login")
```

6. Set the properties for the login screen with the ***resizable()*** and ***configure()*** functions. We will also create a label with the **Label()** class to display on the login screen -

```
self.login.resizable(width=False, height=False)
self.login.configure(width=400, height=300)
self.pls = Label(self.login,
                 text = "Please login to continue",
                 justify = CENTER,
                 font = "Helvetica 14 bold")
```

7. Use the ***place()*** function on the label to place it on the login window -

```
self.pls = Label(self.login,
                 text = "Please login to continue",
                 justify = CENTER,
                 font = "Helvetica 14 bold")
self.pls.place( relheight = 0.15,
                relx = 0.2,
                rely = 0.07)
```

8. Create a ***GUI()*** object so that the function can run when the ***client.py*** is started from the cmd/terminal -

```
class GUI:
    def __init__(self):
        self.Window = Tk()
        self.Window.withdraw()

        self.login = Toplevel()
        self.login.title("Login")

        self.login.resizable(width=False, height=False)
        self.login.configure(width=400, height=300)

        self.pls = Label(self.login,
                         text = "Please login to continue",
                         justify = CENTER,
                         font = "Helvetica 14 bold")
        self.pls.place( relheight = 0.15,
                        relx = 0.2,
                          rely = 0.07)

g = GUI()
```


9. Create a label for the name field on the login window and place it on the screen -

```
self.labelName = Label(self.login,
                       text = "Name: ",
                       font = "Helvetica 12")
self.labelName.place(relheight = 0.2,
                       relx = 0.1,
                       rely = 0.2)
```

10. Add an ***Entry()*** widget next to the label so that the user can input their nickname in the login window. Make sure to use a ***focus()*** function on it, so that as soon as the login window opens, it will keep the input field selected so the user can directly enter their nickname -

```
self.entryName = Entry(self.login,
                       font = "Helvetica 14")
self.entryName.place(relwidth = 0.4,
                       relheight = 0.12,
                       relx = 0.35,
                       rely = 0.2)
self.entryName.focus()
```

11. Create a button for login, and place it on the login screen -

```
self.go = Button(self.login,
                 text = "CONTINUE",
                 font = "Helvetica 14 bold",
                 command = self.goAhead(self.entryName.get()))
self.go.place(   relx = 0.4,
                 rely = 0.55)
```

12. Add the ***lambda*** keyword to the ***command*** attribute, to convert it into a lambda function. Currently, with the ***lambda*** function not added, the ***self.goAhead()*** function will directly execute as soon as the app is opened, without the button being clicked. The lambda function is similar to an arrow function in JavaScript. It is used to create a one line function.

```
self.go = Button(self.login,
                 text = "CONTINUE",
                 font = "Helvetica 14 bold",
                 command = lambda: self.goAhead(self.entryName.get()))
```

13. Add the ***mainloop()*** function to the main window - ***self.Window*** - that is a ***Tk()*** object since it is our main window and ***self.login*** created with ***Toplevel()*** is just a top level window -

```
self.go.place(  relx = 0.4,
                rely = 0.55)

self.Window.mainloop()
```

14. Create the **goAhead()** function in the class. It takes the **name** as an argument, along with the keyword **self**, since this function belongs to the class. It will destroy the login window with the **destroy()** function. Save the name into a variable called **self.name**. Also, since now we have our nickname of the user, we can start receiving messages, so create a thread that starts the ***receive()*** function -

```
def goAhead(self, name):
    self.login.destroy()
    self.name = name
    rcv = Thread(target=self.receive)
    rcv.start()
```

15. Take the **receive()** function that we commented earlier and bring it inside the ***GUI*** class.

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

```
def goAhead(self, name):
    self.login.destroy()
    self.name = name
    rcv = Thread(target=self.receive)
    rcv.start()

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

16. Add the keyword **self** as an argument to it -

```
def receive(self):
    while True:
        try:
```

17. Change the line of code where we were sending the nickname to the server. This time, instead of sending **nickname,** we will be sending **self.name** and also instead of printing the message, just put a **pass** statement there. We can deal with it later when our chat interface is ready -

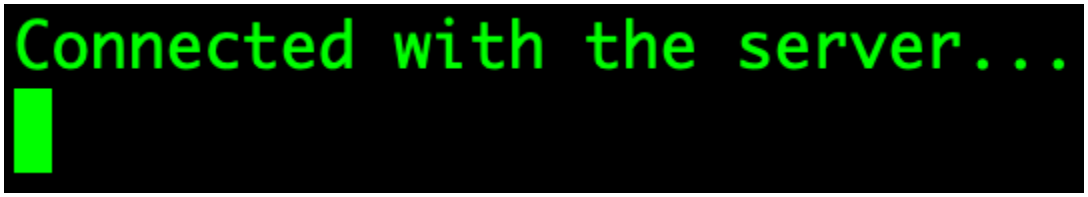
```
try:
    message = client.recv(2048).decode('utf-8')
    if message == 'NICKNAME':
        client.send(self.name.encode('utf-8'))
    else:
        pass
except:
    print("An error occured!")
    client.close()
    break
```


18. Run the **server.py** and the **client.py** in separate command prompts/terminals -

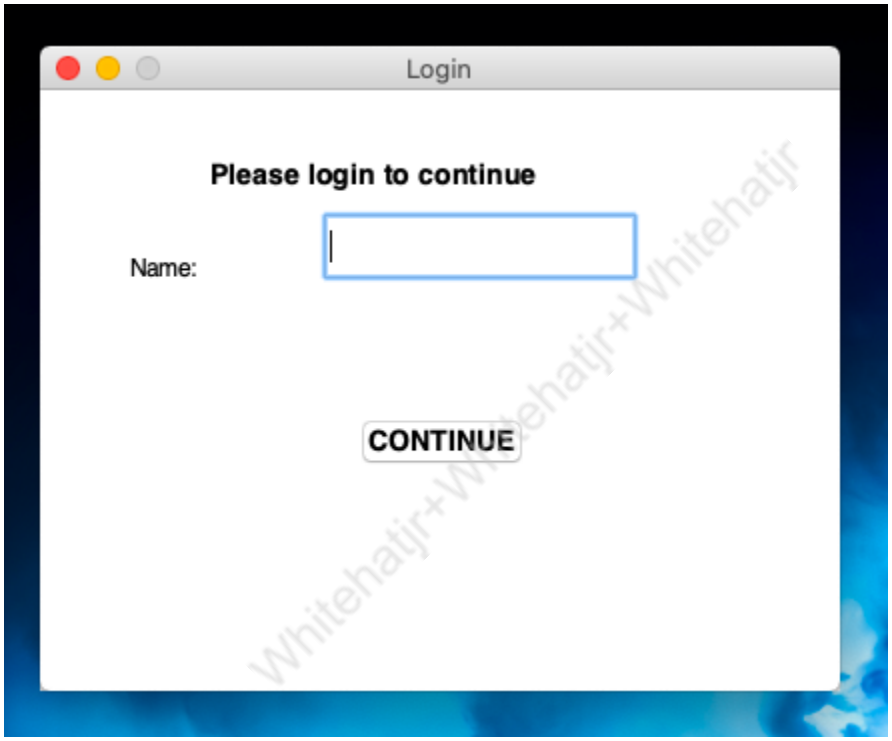
server.py

```
Server has started...
 joined!
```

client.py



19. The login window opens up and looks like -



What's NEXT?

In the next class, we will complete the chat interface of this app and have a fully functional GUI based chat applications

Expand Your Knowledge:

Explore more about Tkinter's library here