Topic	GUI BASED CHAT APP	
Class Description	Students will add a GUI to the client app they built and deep dive into Tkinter library	
Class	C-202	
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

Question	Answer

TEACHER-LED ACTIVITY - 10mins

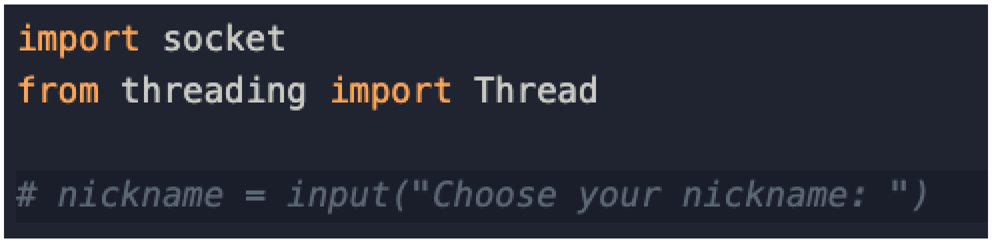
Teacher Initiates Screen Share

ACTIVITY

- **Creating the Login Screen into the Chat Application**

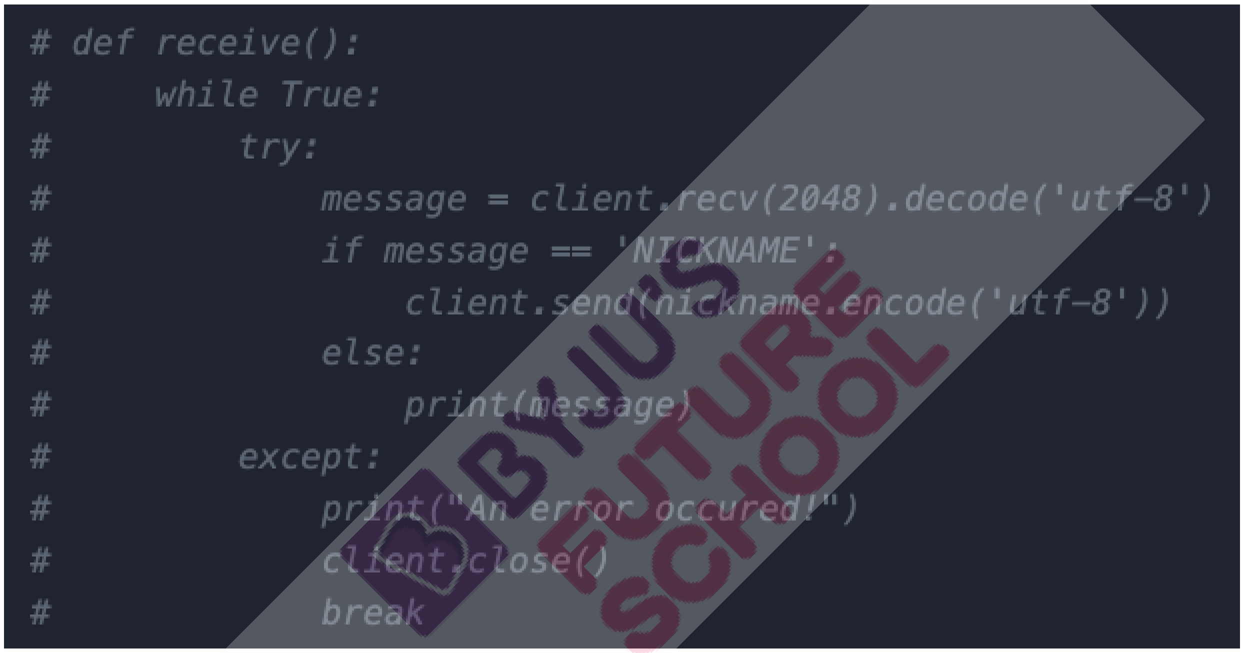
Teacher Action	Student Action
In the last class, we got started with the Tkinter library in Python and built an application to calculate the Body Mass Index (BMI). Any doubts from the last session? *Teacher resolves the query of the student (if any)* Great! Can you tell me what Tkinter is?	**ESR:** Varied! **ESR:** It is a Python library to build GUI for desktop applications!
Great! Now do you remember the chat application we built that used to run on the command prompt/terminal? It consisted of 2 parts - 1. Server	

2. Client Can you tell me what a server is? And can you tell me what a client is?	**ESR:** Server is the one that works behind the scenes. It's like the back-end of the app. **ESR:** Client is the program that the user uses. It's like the front-end of the app.
Awesome! Now since server is the backend part, we really don't have to change it but to add GUI to our chat app using Tkinter, we will have to change the ***client.py***. Let's discuss the UI first, before we start building it. Our socket app first used to take the nickname from the user, before starting the threads that ***receive()*** and ***write()*** messages.	
To break the GUI down, we can have 2 different screens - 1. To take the nickname of the user 2. The chat interface, once the user logs into it	
Let's get into our ***client.py*** and start with the Nickname screen first! *Teacher clones the repository from Teacher Activity 1* This is the same code from C-200!	*Student clones the repository from Student Activity 1*
Now let's start working on our ***client.py*** The first thing to notice is that we are taking the nickname from the user. Let's comment out that code, since we now want to take the input from the GUI -	

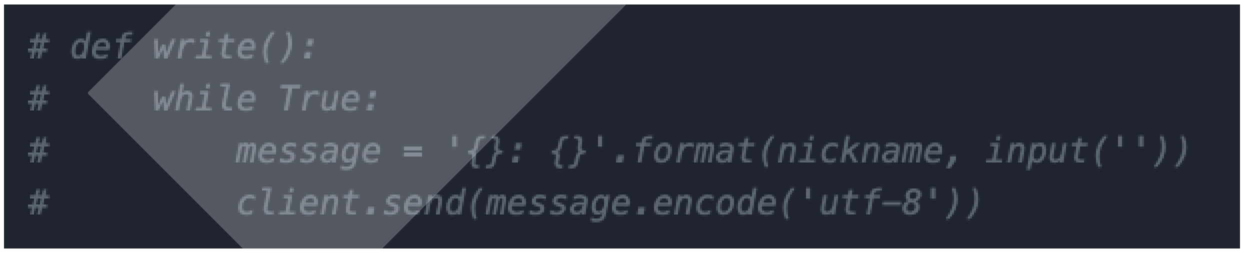


```python
import socket
from threading import Thread

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

Also, let’s comment out our ***receive()*** and the ***write()*** functions for now -

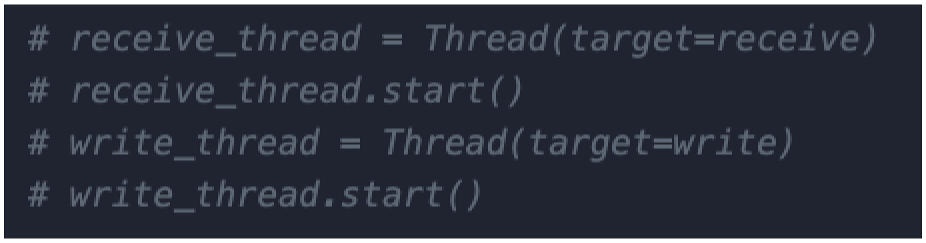


```python
# def receive():
#     while True:
#         try:
#             message = client.recv(2048).decode('utf-8')
#             if message == 'NICKNAME':
#                 client.send(nickname.encode('utf-8'))
#             else:
#                 print(message)
#         except:
#             print("An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



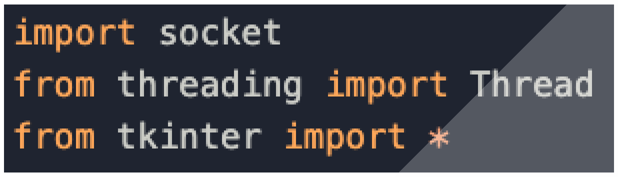
```python
# def write():
#     while True:
#         message = '{}: {}'.format(nickname, input(''))
#         client.send(message.encode('utf-8'))
```

And also the Threads that are starting these functions -



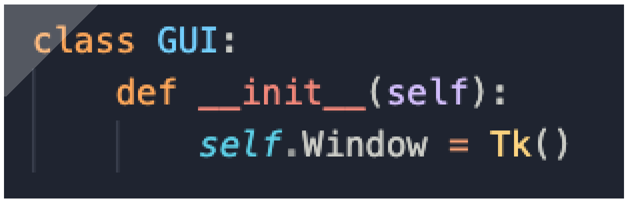
```python
# receive_thread = Thread(target=receive)
# receive_thread.start()
# write_thread = Thread(target=write)
# write_thread.start()
```

Okay! Now let's import everything from the ***tkinter*** module -



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

Now, we know that Python is an Object Oriented Programming language (OOP). To create an object in python, what can we use? That's right. Now, Just how class components had a constructor in React Native, Python has an ***__init__()*** method which initialises the object. Also, instead of ***this*** keyword widely used in JavaScript to refer to self, Python uses the keyword ***self***. Let's say that we create a class called ***GUI***, which will be our main ***tkinter*** window. This class can then handle the receiving and writing of the messages too! Let's create the class -	**ESR:** Classes



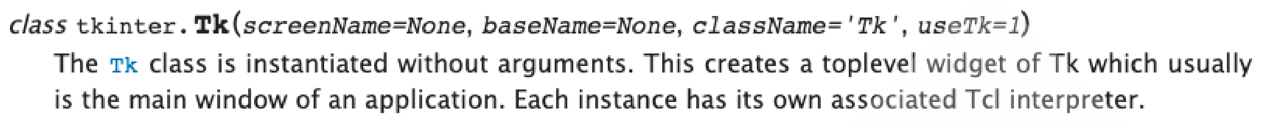
```python
class GUI:
    def __init__(self):
        self.Window = Tk()
```

Now here, we have a class called ***GUI***, which has an initialisation function called

__init__(). This function takes an argument self because it is initialising the object itself.

In this ***init()*** function, we are creating a variable for ***self*** called ***Window***, which is again an object of the ***Tk()*** class.

As the documentation says, ***Tk()*** is used to create the main window of the app.



class tkinter.**Tk**(*screenName=None*, *baseName=None*, *className='Tk'*, *useTk=1*)

The Tk class is instantiated without arguments. This creates a toplevel widget of Tk which usually is the main window of an application. Each instance has its own associated Tcl interpreter.

You can find out more in the documentation from Teacher Activity 2 (*Student Activity 2 for the student)*

Now, as soon as a class object for ***GUI*** is created, we want to display the window where the user can enter their nickname for the app.

If this is the first thing we want to do as soon as the program is run, it only makes sense to do it all in the ***__init__()*** function, since that's the function that is called when any object for ***GUI*** is created.

Let's think about how we can do that now.

To begin first, we have 2 screens. The login screen where the users can enter their nicknames, and the main chat interface screen.

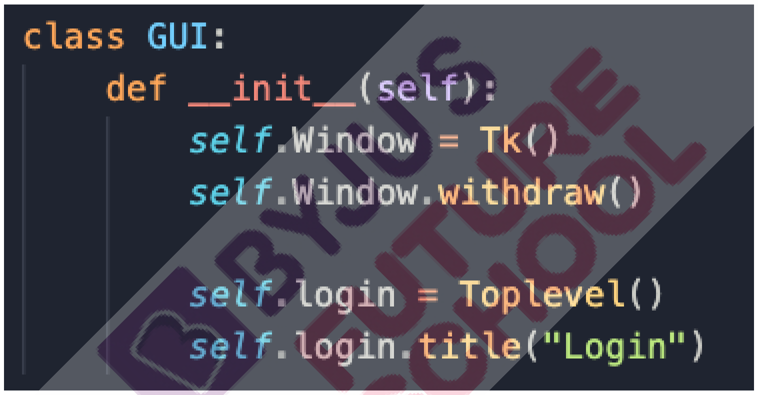
Now the way we can do this is, we can create 2 separate windows for it. One window will be for the login screen, while the other one could be for the chat interface.

The one that we just created with the ***Tk()*** class can be used as the main chat interface screen, but it will come later on when we want to show the user the chat interface after they enter their nickname.

We can hide this window and create a top level window on top of it with the ***Toplevel()*** class for the login screen but the question is, what's the difference between ***Tk()*** and

ESR:

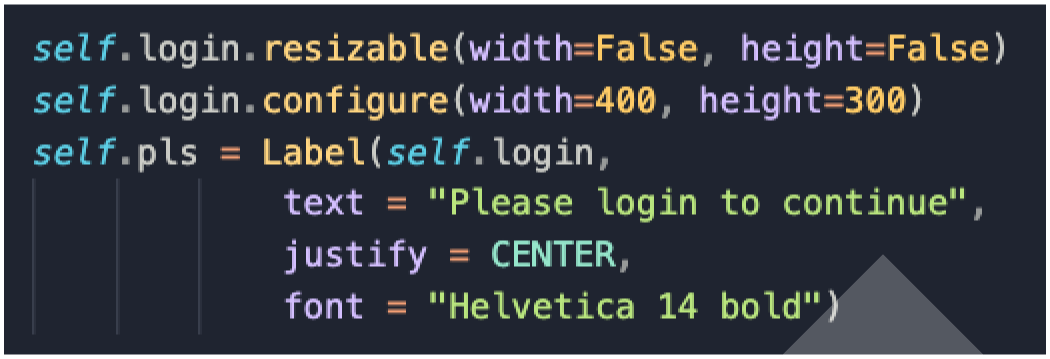
Toplevel() if they both are used to create windows? Well, **Tk()** is used to create the main window for the application. **Toplevel(),** on the other hand, can be used to create a top level window for any extra functionality. The difference is really about the features and things we can do with both the windows. Since our main functionality is the chat interface, we can keep the window we initialised in **self.Window** for later and create another top level window for login first. Let's do that -	Varied!



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

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

Here, if you note, we are using a ***withdraw()*** function on the ***Tk()*** object that we created. This ***withdraw()*** function is used to hide, or withdraw the window temporarily from the user's view.

Next, we are creating a new ***Toplevel()*** object called ***self.login***, and we are giving it a title with the ***title()*** function called **"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")
```

Next, we can define that we don't want our top level window resizable by using the ***resizable()*** function and passing the arguments of ***width*** and ***height*** as ***False***.

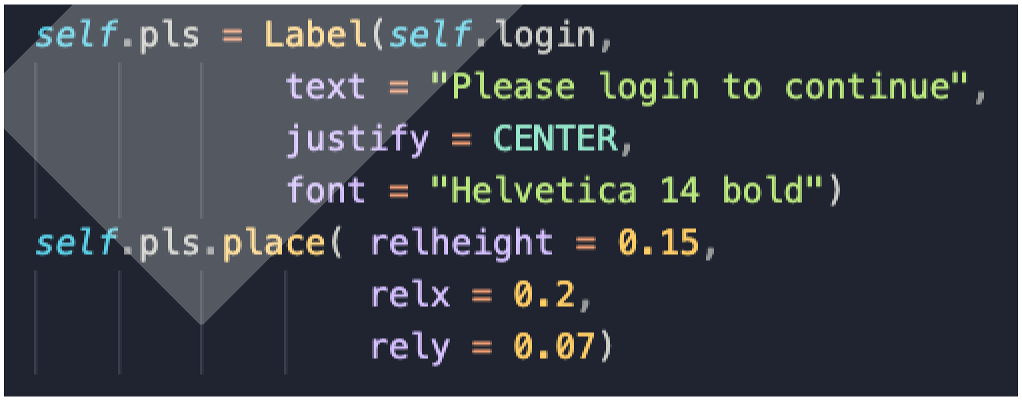
We can also configure it's width and height with the ***configure()*** function and pass the arguments of ***width*** and ***height***. We have the ***width as 400*** and ***height as 300***.

Finally, we create a label with the ***Label()*** function called ***self.pls***

In the ***Label()*** function, our first argument is the window in which we want to display it, in our case, ***self.login***

Next, we define it's text with the ***text*** argument, we justify it to ***CENTER*** with the ***justify*** argument and define it's font with the ***font*** argument.

Note that we are still to define its position on the window! We can do it with the ***place()*** function -



```
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)
```

Here, you can see that we have used the arguments ***relheight, relx*** and ***rely*** in the ***place()*** function.

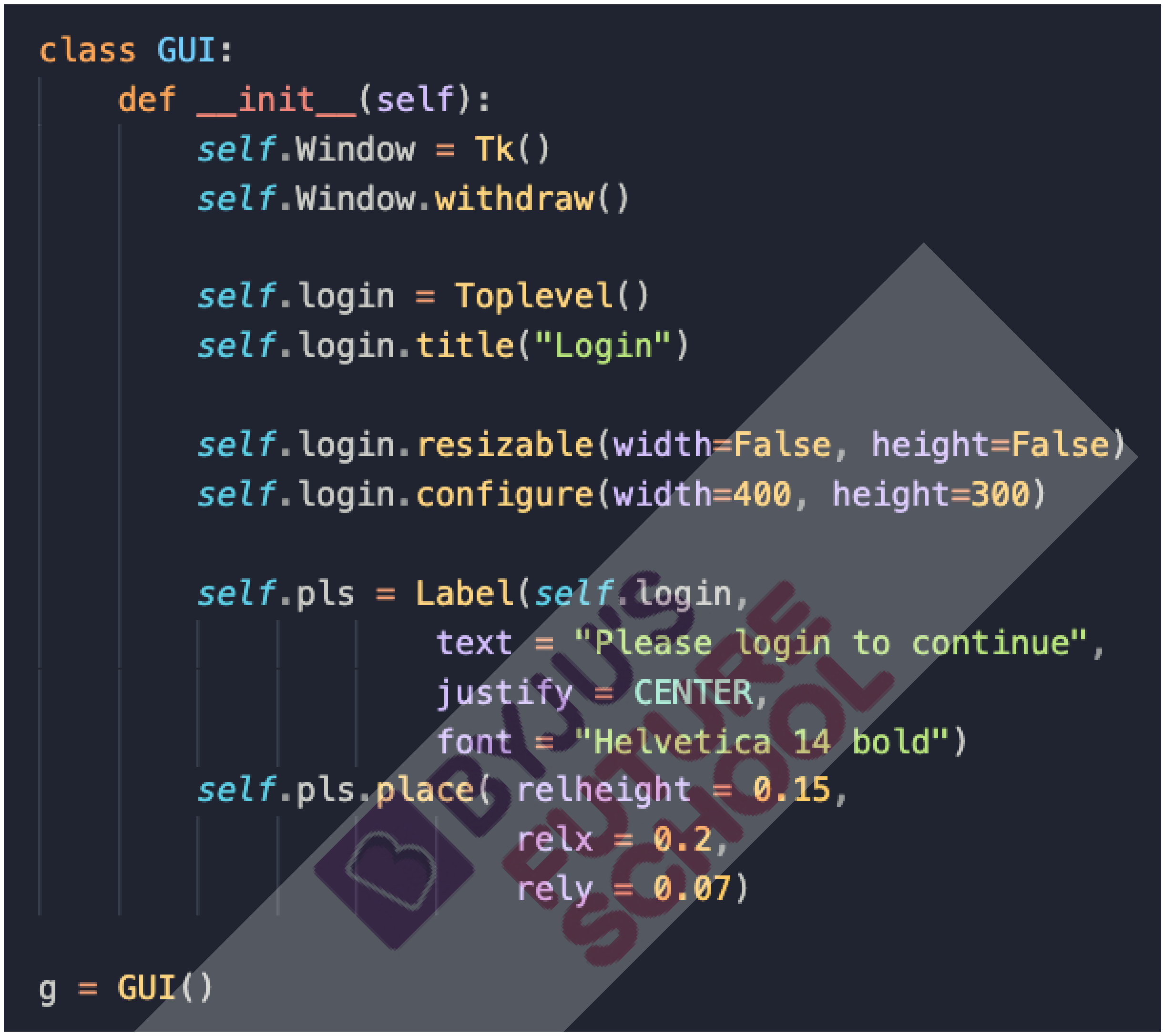
relheight means the relative height of the widget, ***Label*** in our case.
relx means the relative x position of the widget and ***rely*** means the relative y position of the widget.

Now, these values are all relative to the parent widget, which in our case, is the top level window itself.

Also, the values for these arguments have been achieved through hit and trial, but that's all about building the front-end.

We often find ourselves doing hit and trial while building games, UIs, etc.

Let's also create a ***GUI*** object at the bottom, so out ***__init__()*** function can run as soon as we run the ***client.py*** -



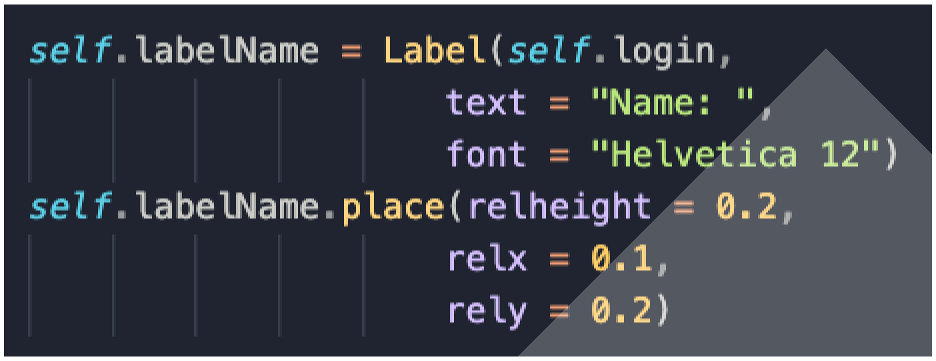
```
class GUI:
    def __init__(self):
        self.Window = Tk()
        self.Window.withdraw()

        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()
```

Okay, now our Login window is going to have a title and a label at the top. Now let's think what else do we need? Excellent! Let's work on the name label and the name entry first!	**ESR:** We need a name label and a name entry widget for users to enter their nickname. We also need a button to submit the nickname and login!

Can you help me out with the code?

Teacher writes the code with the student's help

Student helps the teacher in writing the code

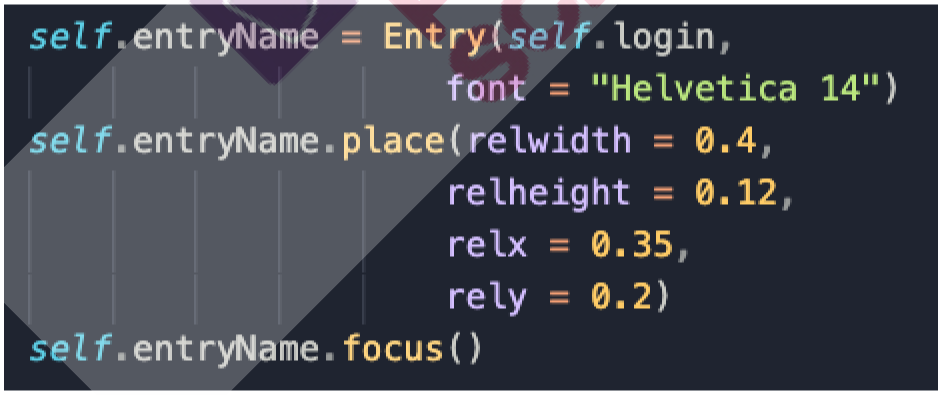


```
self.labelName = Label(self.login,
                       text = "Name: ",
                       font = "Helvetica 12")
self.labelName.place(relheight = 0.2,
                     relx = 0.1,
                     rely = 0.2)
```

Here, we create a label for the name with the ***Label()*** class, and give the window in which we want it to appear (login window), the text that it will have and the font style.

We are then using the ***place()*** function with ***relheight, relx*** and ***rely*** to place this label into our window.

Similarly, we create an ***Entry()*** for taking the user input -



```
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()
```

Here, the ***focus()*** function on the ***Entry()*** object is to ensure that the input field is selected as soon as the window is opened by default, so the user doesn't have to click on it specifically and can directly enter their nickname.

We've achieved quite a lot! Now all we need is a button that will enable the user to continue to the main chat interface, and also, we will need a function that is called when the

button is clicked to handle the event! It's your turn from here.	

Teacher Stops Screen Share

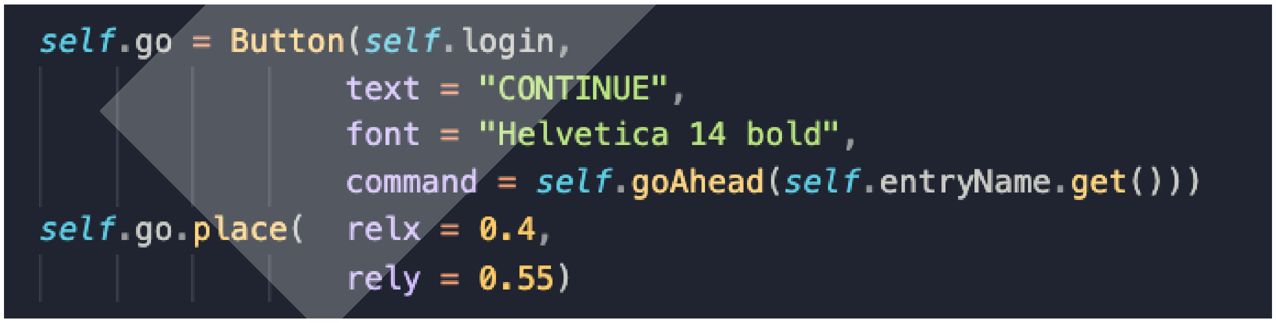
STUDENT-LED ACTIVITY - 20 mins

- **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

- **Adding button to the login window**
- **Adding the button handler function**

Teacher Action	Student Action
Guide the student to get the boilerplate code from Student Activity 3	*Student clones the code from Student Activity 3*
Alright, let's start by adding the button into our login window - *Teacher helps the student in writing the code*	**ESR:** *Student writes the code*



```
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)
```

Hare, a button object is created with the ***Button()*** class and is saved in a variable called ***self.go***
Now this ***Button()*** class took the first argument, the window in which it should be displayed, then it took the ***text*** and the ***font*** it should use and 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.

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

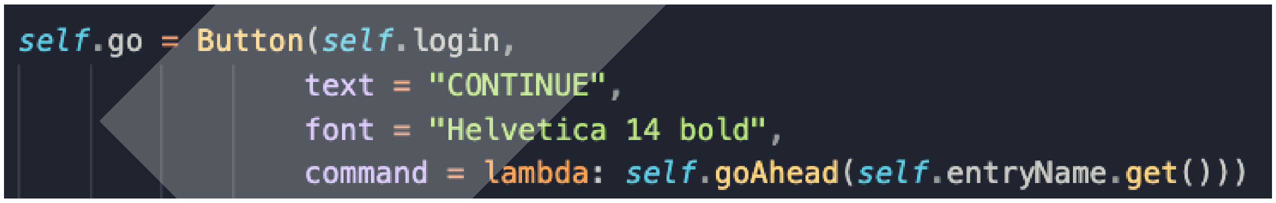
This function is required to be a part of the class ***GUI***, so we are using the keyword ***self*** with the name of the function ***goAhead()*** as ***self.goAhead()***. Now, this function will take one argument, which will be the nickname the user entered. We can get it from the entry widget we created just now with the ***get()*** function.

Let's worry about this ***goAhead()*** function that we have to create later, and use the ***place()*** function on the button to place it on the login screen.

Now this one more thing, we need to be careful about while using the ***command*** argument in the button is that, this function contains the ***()*** with an argument of the name inside it, which will call it as soon as the ***GUI*** class object is initialised when we run the program.

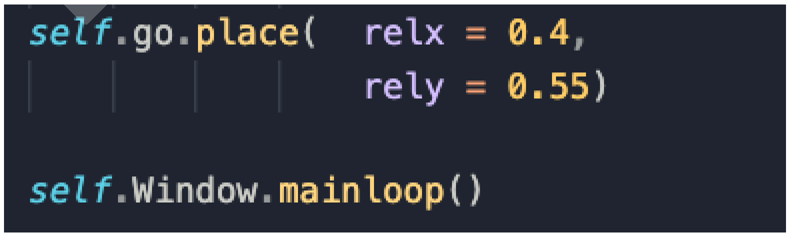
This means that the ***self.goAhead()*** function will be called as soon as the program starts running, before waiting for the button click.

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



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

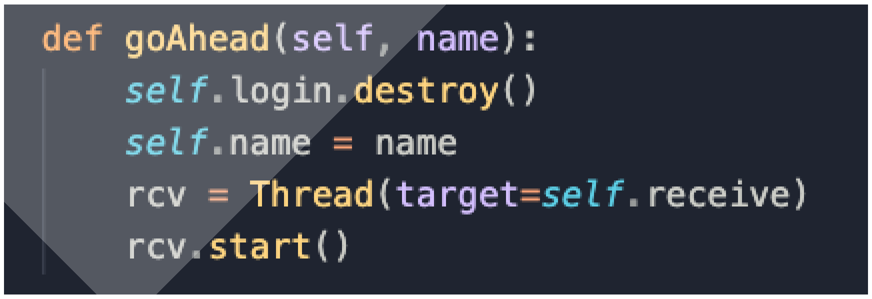
Finally, we will have -



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

self.Window.mainloop()
```

This is because we have only withdrawn the main window from the display, but this is still our main window. The top level login window we created with the ***Toplevel()*** class does not require the ***mainloop()*** method to run, since it is running on top of the main window we created with the ***Tk()*** class saved in ***self.Window***, which requires the ***mainloop()*** method to run the whole program.

Alright, now the login window is done. We still need to work on the **goAhead()** function which handles the button click. Let's think about it for a moment! The first thing we want to do here is to destroy the top level login window, because it's job is done. Also, if you think about the flow of the client in the previous ***client.py***, we used to take the nickname at the very beginning of the program. Here, when this **goAhead()** function is called, we will have the nickname, which means we can at least start receiving the messages, since our server first looks for the nickname as soon as our client makes a request! We can again use a thread for it! Let's write the code. *Teacher helps the student in writing the code*	*Student writes the code*



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

Here, we have defined a function called ***goAhead()*** inside the class, just like the ***__init__()*** function, with 2 arguments -

1. ***self*** - Because this function belongs to the client itself
2. ***name*** - Containing the nickname that the client chose

We are also saving the nickname user created to a variable called ***self.name***

Inside this function, the first thing that we do is to destroy the top level login window we created with the ***destroy()*** function.

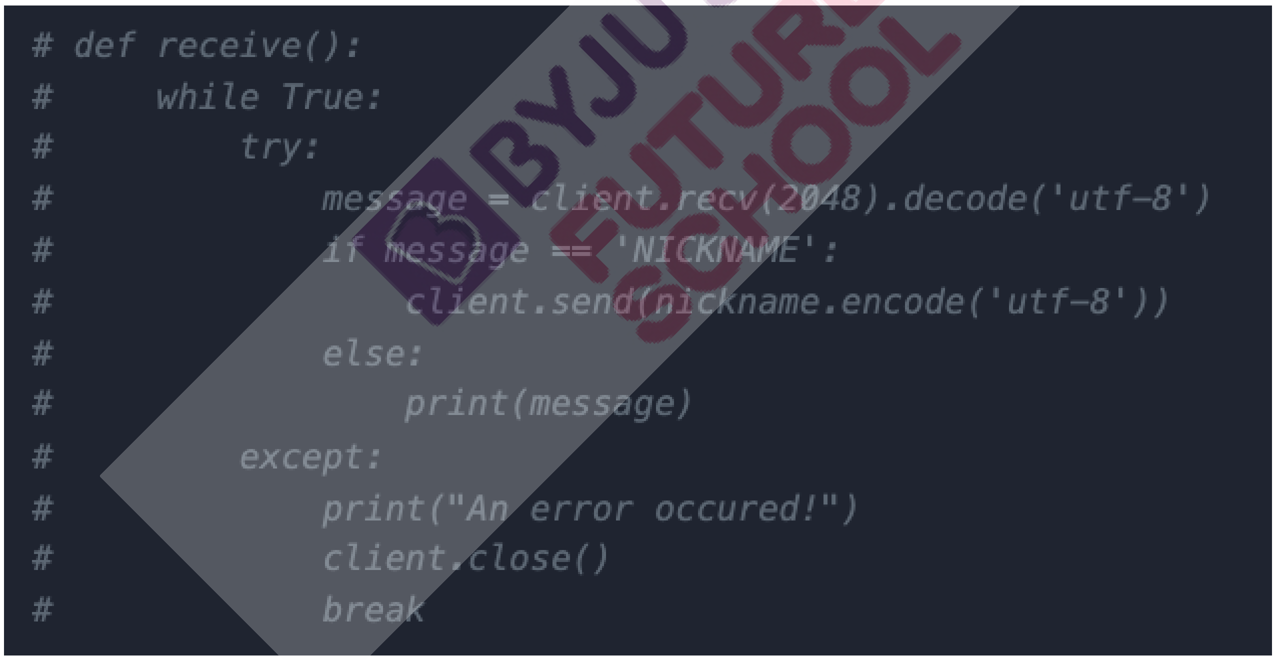
Next, we create a new ***Thread()*** and call a function called ***self.receive*** inside it by setting it as the ***target***. We then start the thread with the ***start()*** function.

Now finally we have the **receive()** function. Do note that this function was there earlier, but now it needs to be there in the ***GUI*** class. Also, our ***nickname*** variable does not exist, but instead, it's ***self.name***

Now let's get that function back -

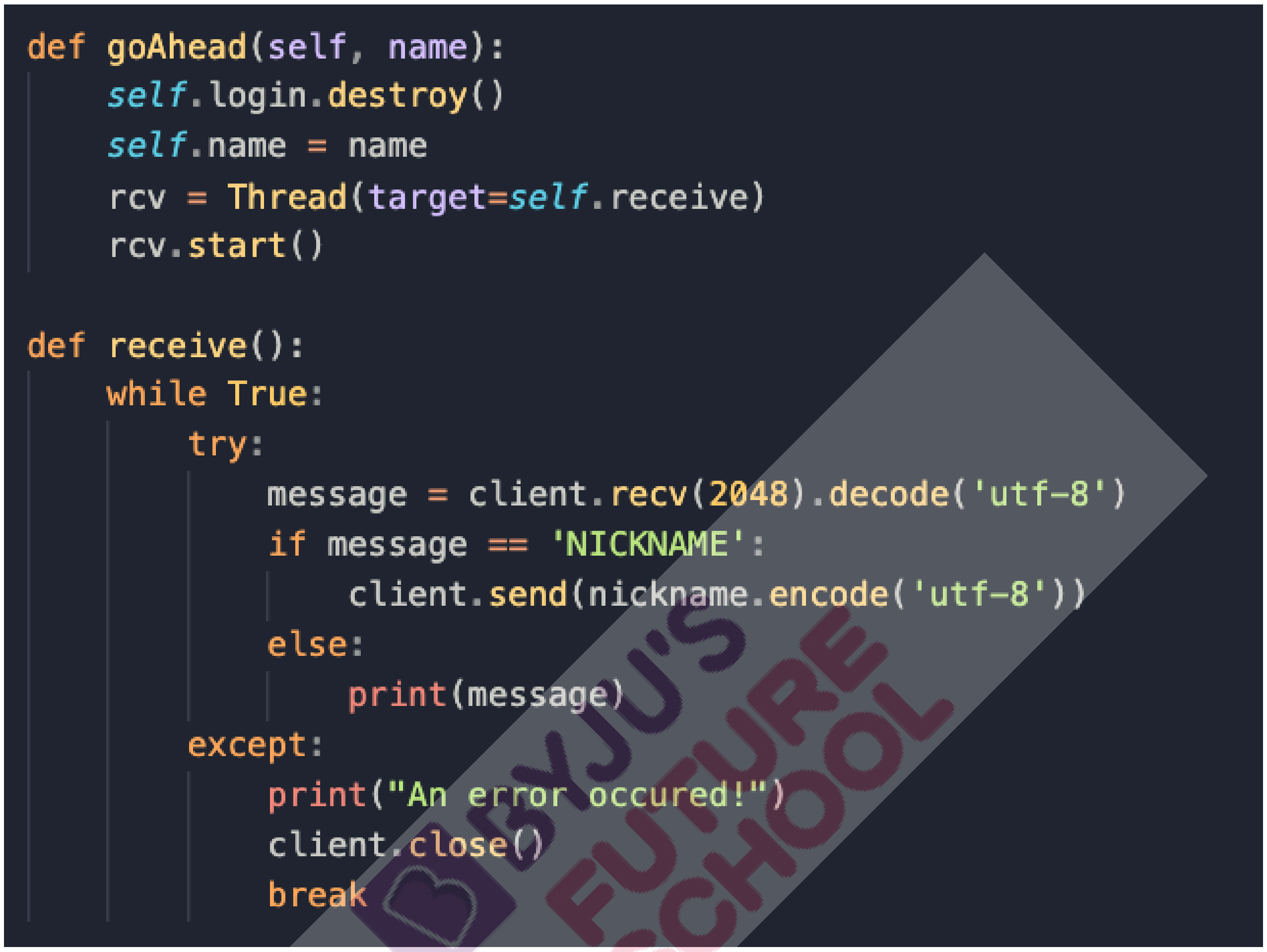
Teacher helps the student in writing the code

Student writes the code



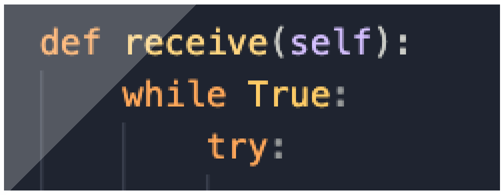
```
# def receive():
#     while True:
#         try:
#             message = client.recv(2048).decode('utf-8')
#             if message == 'NICKNAME':
#                 client.send(nickname.encode('utf-8'))
#             else:
#                 print(message)
#         except:
#             print("An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

We will first copy the function that we commented out and paste it into the class and uncomment it -



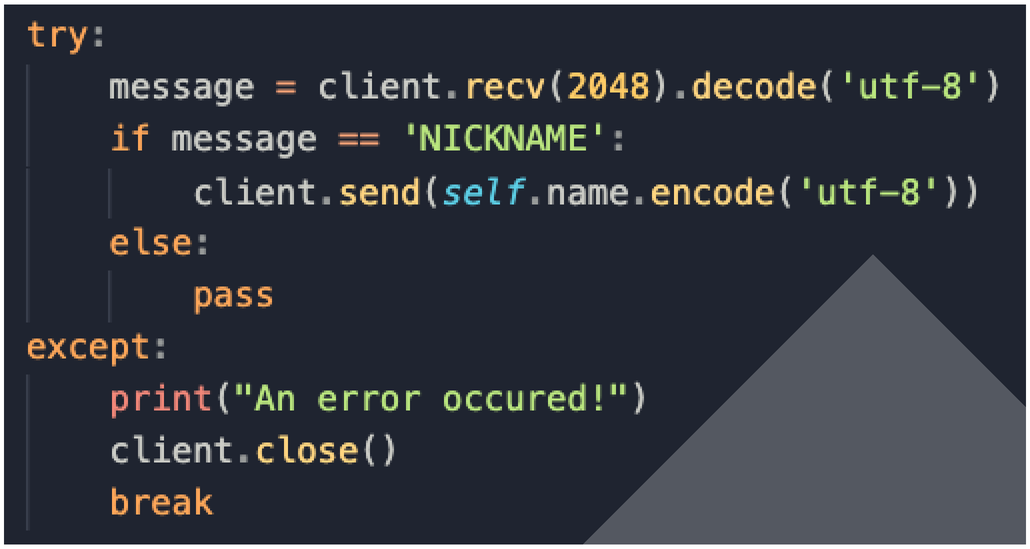
```
def goAhead(self, name):
    self.login.destroy()
    self.name = name
    rcv = Thread(target=self.receive)
    rcv.start()

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
```

We will then add the keyword ***self*** to the function as an argument, since now it's a function of the class ***GUI*** -



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

We will also change the code where we send the nickname to the server and instead of printing the message, let's just have a ***pass*** statement for now, because now we will want to display the messages in the chat interface that we are yet to build -

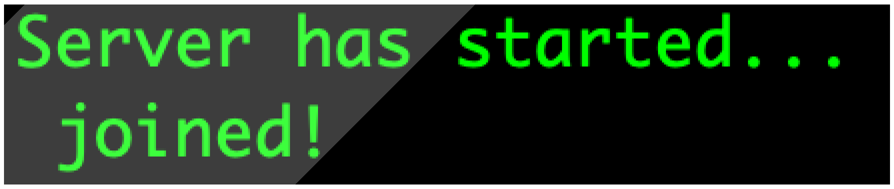


```
try:
    message = client.recv(2048).decode('utf-8')
    if message == 'NICKNAME':
        client.send(self.name.encode('utf-8'))
    else:
        pass
except:
    print("An error occured!")
    client.close()
    break
```

You will notice that we are now using ***self.name*** instead of ***nickname*** opposed to earlier.

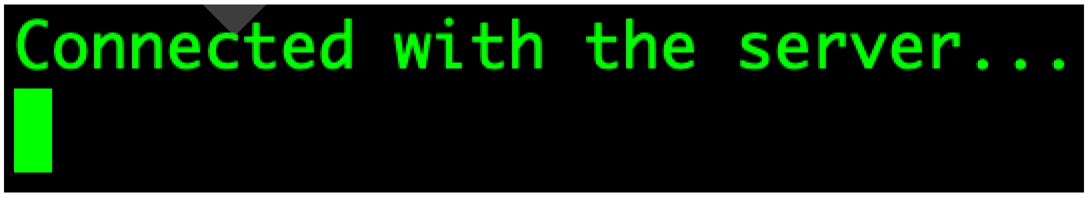
Now let's try running the program. Run the ***server.py*** in a separate command prompt/terminal, and run the ***client.py*** in a separate command prompt/terminal ***Teacher helps the student in running the files***	***Student runs the files***

server.py in terminal/cmd looks like -



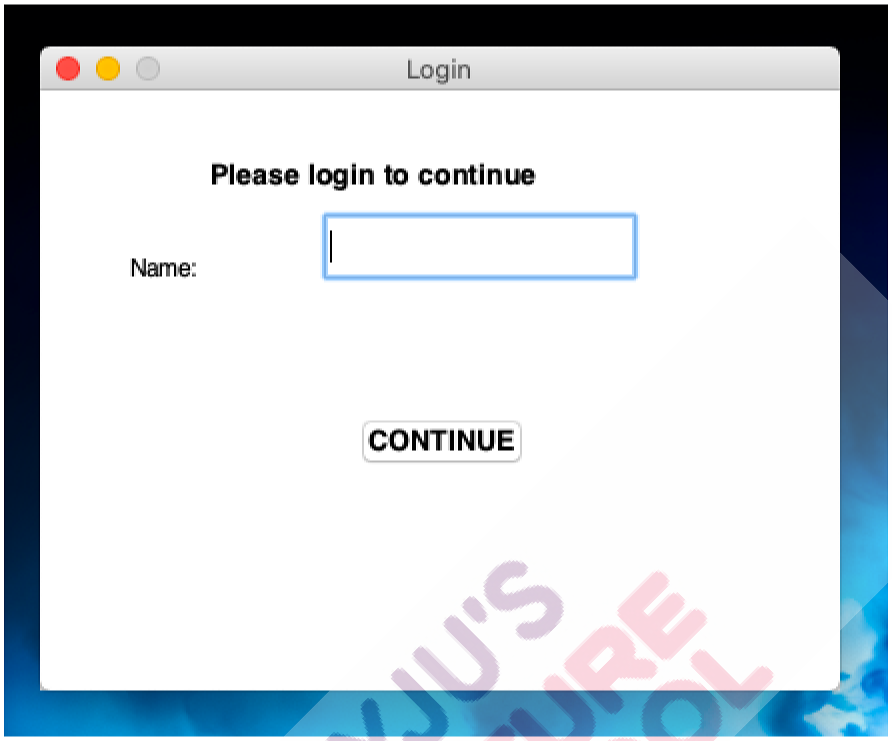
```
Server has started...
 joined!
```

client.py in the terminal/cmd looks like -



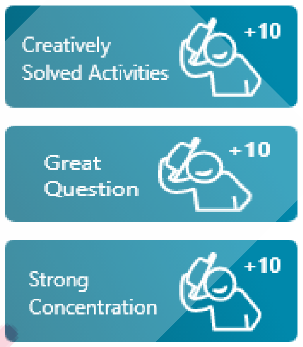
```
Connected with the server...
```

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



Amazing! Our first part of the app is complete! Now, in the next class, we will be working on the chat interface of the app.	
Teacher Guides Student to Stop Screen Share	
WRAP UP SESSION - 5 Mins	
Quiz time - Click on in-class quiz	
Question	**Answer**
End the quiz panel	
<u>FEEDBACK</u> ● **Appreciate the students for their efforts in the class.**	

BYJU'S FUTURE SCHOOL

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

● What aspects of the class helped me? What did I find difficult?	

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	C-200 Reference Code	https://github.com/pro-whitehatjr/C200_TeacherReference
Teacher Activity 2	Tkinter Library	https://docs.python.org/3/library/tkinter.html
Teacher Activity 3	Reference Code	https://github.com/pro-whitehatjr/PRO-202_Solution
Student Activity 1	C-200 Reference Code	https://github.com/pro-whitehatjr/C200_TeacherReference
Student Activity 2	Tkinter Library	https://docs.python.org/3/library/tkinter.html
Student Activity 3	Boilerplate Code	https://github.com/pro-whitehatjr/PRO-202_Boileplate