| Topic | File Sharing App - 3 | |
|---|---|---|
| **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-210** | |
| **Class time** | **45 mins** | |
| **Goal** | ● Understand about FTP<br>● Making functions for GUI | |
| **Resources Required** | ● Teacher Resources:<br>○ Laptop with internet connectivity<br>○ Earphones with mic<br>○ Notebook and pen<br>○ Visual Studio Code<br><br>● Student Resources:<br>○ Laptop with internet connectivity<br>○ Earphones with mic<br>○ Notebook and pen<br>○ Visual Studio Code | |
| **Class structure** | **Warm-Up**<br>**Teacher - led Activity 1**<br>**Student - led Activity 1**<br>**Wrap-Up** | **10 mins**<br>**10 mins**<br>**20 mins**<br>**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 yview property?<br><br>A. To move content from top to bottom<br>B. To move content from left to right<br>C. To move content from right to left<br>D. None of the above | **A** |
| What can we do to make the GUI button work?<br><br>A. Call the button function when the command is issued<br>B. Call the object when the command is issued<br>C. Click the button<br>D. None of the above | **A** |
| **TEACHER-LED ACTIVITY - 10mins** | |
| **Teacher Initiates Screen Share** | |
| **ACTIVITY**<br><br>• **Write function for Connect Button**<br>• **Display the message according to the button click** | |
| **Teacher Action** | **Student Action** |
| Okay, so you remember what we did in the last session<br><br>Great!<br><br>Any doubts from last session?<br><br>*The teacher clarifies doubts (if any)*<br><br>How about moving on to the next part? | **ESR**<br>FTP_GUI<br><br><br><br><br>**ESR:**<br>Yes! |

| | |
|---|---|
| Let's move on to the third part of the FTP application?<br><br>You remember what FTP is?<br><br>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.<br><br>As we discussed in the last lesson, we set up connections between client and server and designed our FTP user-interface.<br><br>However, we need functionality at the backend in order to make everything work.<br><br>Today we will write functions to handle our buttons. Buttons act like events, when we click them, they trigger an output.<br><br>Today we will cover functionality of connect button, disconnect button.<br><br>All these buttons need functions at client side and to handle these functions we need to write the code for server side as well. | **ESR**<br>**Yes!** |
| *Teacher download the boilerplate code from [Teacher Activity 1]* | *Student download the repository from [Student Activity 1]* |
| Now that the refresh button is working, we need a way to send messages from the client to the server. What information should be displayed and how the client's data should be fetched and how much data needs to display.<br><br>***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*

```
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 start writing the code from here*<br><br>Now that we are done with our Refresh button,which display showlist and send info to server using **receiveMessage()**<br><br>let's code the Connect button<br><br>When we click on the connect button, what will happen?<br><br>When we click connect, it will connect us to the other client so we can begin chatting<br><br>That's right! | **ESR**<br>The client will be connected to the other's client |
| Create a function for the connect button and call it when the Connect button is clicked<br><br>The connect button must be enabled by selecting a user from the list box, therefore let's write the function to achieve this .<br><br>Make function ***connectWithClient()*** Use global variable SERVER, listbox<br><br>● Create a variable text to store selected users from the list which we can achieve by listbox.get anchor value function. Listbox.get anchor() will help to get the selected user which has been selected by the client.<br><br>● Listbox data is separated by a **split()** method which will separate the strings using colon when they are retrieved from the list box.<br><br>● Variable msg is created to store the client name chosen from a list box when the client connection is made.Then send the message in encoding format to | |

the server

```
def connectWithClient():
    global SERVER
    global listbox

    text=listbox.get(ANCHOR)
    list_item = text.split(":")
    msg="connect "+list_item[1]
    SERVER.send(msg.encode('ascii'))
```

This function needs to be called at the user-interface in order for **"Connect"** Button to work

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

We need to create server-side functions to deal with this information as soon as the server receives the client request.

In order to connect a client with another client, the ***connectClient()*** function is used.

- It takes three arguments: message as entered by user in format "connect entered_client_name", client is the sender client socket and client_name will contain the name of the client sending the connection request.
- other_client_socket is the socket connection which will be made with the recipient client.
- Global variable clients is again the connected clients list stored on the server.
- When Entered_client_name is fetched from the user

| | |
|---|---|
| message then server checks if this client is connected to the server or not and also if the recipient client is available to chat or is connected to some user.<br><br>• If the client is available, a successful message is sent to both clients, else the message is sent to the user that is already connected to some other client. | |

```
def connectClient(message, client, client_name):
    global clients

    entered_client_name = message[8:].strip()
    if(entered_client_name in clients):
        if(not clients[client_name]["connected_with"]):
            clients[entered_client_name]["connected_with"] = client_name
            clients[client_name]["connected_with"]  = entered_client_name

            other_client_socket = clients[entered_client_name]["client"]

            greet_message = f"Hello, {entered_client_name} {client_name} connected with you !!!"
            other_client_socket.send(greet_message.encode())

            msg = f"You are successfully connected with {entered_client_name}"
            client.send(msg.encode())
        else:
            other_client_name = clients[client_name]["connected_with"]
            msg = f"You are already connected with {other_client_name}"
            client.send(msg.encode())
```

| | |
|---|---|
| Now as we know, we have various types of buttons to use, and depending on the button's function, it will display a message in the text area.<br><br>Can you tell me how many buttons?<br><br>Let's create a function ***handleMessage()***<br><br>It will check which button is clicked, if refresh is clicked call the function **handleshowclient()**<br><br>If connect button is clicked call the function ***connectClient()*** | **ESR**<br>Refresh, Connect, disconnect |

| | |
|---|---|
| If disconnect button is clicked call the function ***disconnectWithClient()***<br><br>As we have not written that function for disconnection i am writing pass here.<br><br>Now your turn to write the disconnect function and call the same in handle message. | |

```
def handleMessges(client, message, client_name):
    if(message == 'show list'):
        handleShowList(client)
    elif(message[:7] == 'connect'):
        connectClient(message, client, client_name)
    elif(message[:10] == 'disconnect'):
        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.**

**ACTIVITY**

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

| Teacher Action | Student Action |
|---|---|
| Now that we have finished the refresh and connect buttons, it's time to write the code for disconnect buttons | *Student clones the code* |

| | |
|---|---|
| *Guide the student to get the boilerplate code from Student Activity 1* | *from Student Activity1* |
| It is necessary to select the user from the list box for this disconnect button to work. Let us create a function that accomplishes this.<br><br>Make a function named ***disconnect()***<br><br>● Create a variable text to store selected users from the list which we can achieve by listbox.get anchor value function. **Listbox.get anchor()** will help to get the selected user which has been selected by the client for disconnection<br><br>● Listbox data is separated by a ***split()*** method which will separate the strings using colon when they are retrieved from the list box.<br><br>Variable msg is created to store the client name chosen from a list box when the client connection is made.Then send the message in encoding format to the server | |

```
def disconnectWithClient():
    global SERVER

    text=listbox.get(ANCHOR)
    list_item = text.split(":")
    msg="disconnect "+list_item[1]
    SERVER.send(msg.encode('ascii'))
```

| | |
|---|---|
| This function needs to be called at the user-interface in order for Connect to work | |

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

To handle this at the server side, we have to make this function at the server side as well.

Let's give name to the function ***disconnectWithClient()***

***disconnectWithClient()*** function is used by a client to disconnect from the client it's connected to.

- This takes the same arguments as ClientConnection(). Just the message format will be different and will be as "disconnect entered_client_name".
- The connected_with attribute will be set to ""(empty string) for both the sender and the recipient client and both the clients will be notified of the same via a message from the server

```
def disconnectWithClient(message, client, client_name):
    global clients

    entered_client_name = message[11:].strip()
    if(entered_client_name in clients):
        clients[entered_client_name]["connected_with"] = ""
        clients[client_name]["connected_with"]  = ""

        other_client_socket = clients[entered_client_name]["client"]

        greet_message = f"Hello, {entered_client_name} you are successfully disconnected with {client_name} !!!"
        other_client_socket.send(greet_message.encode())

        msg = f"You are successfully disconnected with {entered_client_name}"
        client.send(msg.encode())
```

The disconnect function has been created for both the server and client, so lets call it on the ***handleMessage()*** function side as well.

*Teacher helps the student in writing the code*

*Student writes the code*

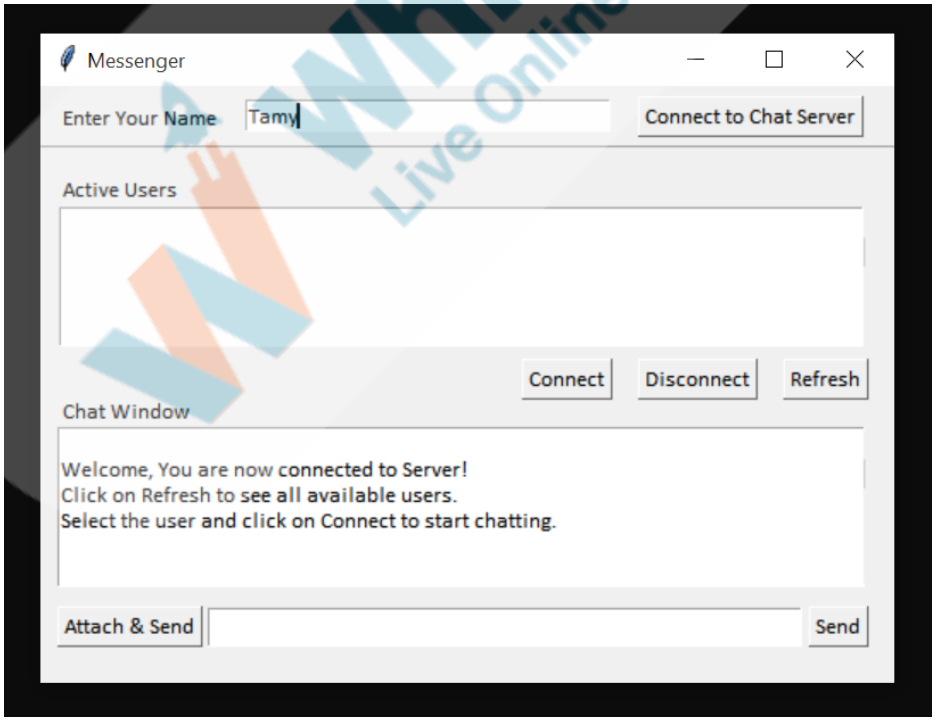
```
def handleMessges(client, message, client_name):
    if(message == 'show list'):
        handleShowList(client)
    elif(message[:7] == 'connect'):
        connectClient(message, client, client_name)
    elif(message[:10] == 'disconnect'):
        disconnectWithClient(message, client, client_name)
```

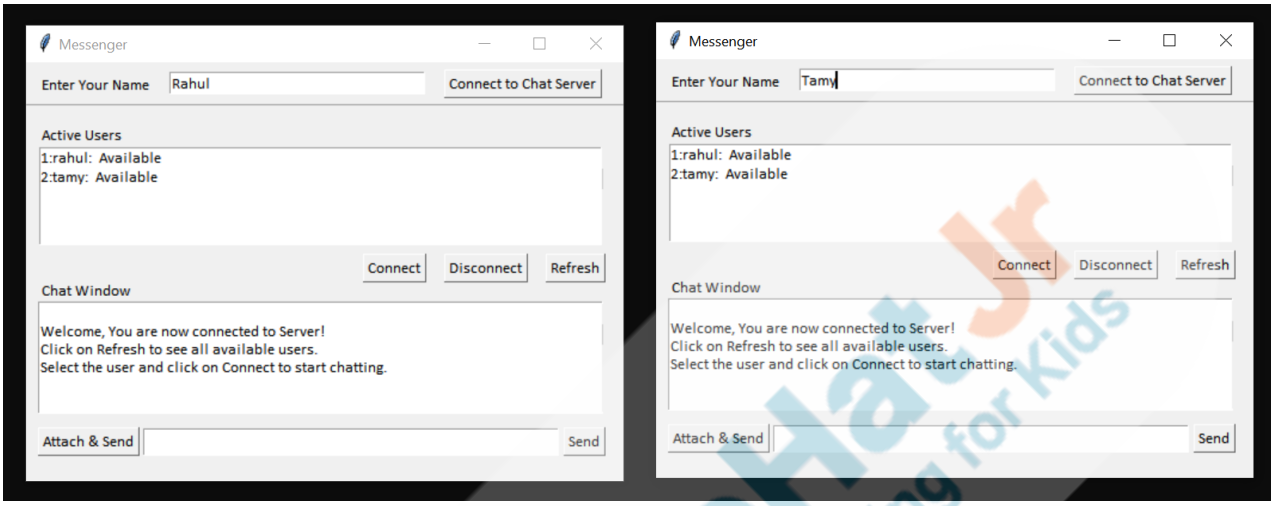
***server.py*** in terminal/cmd looks like -

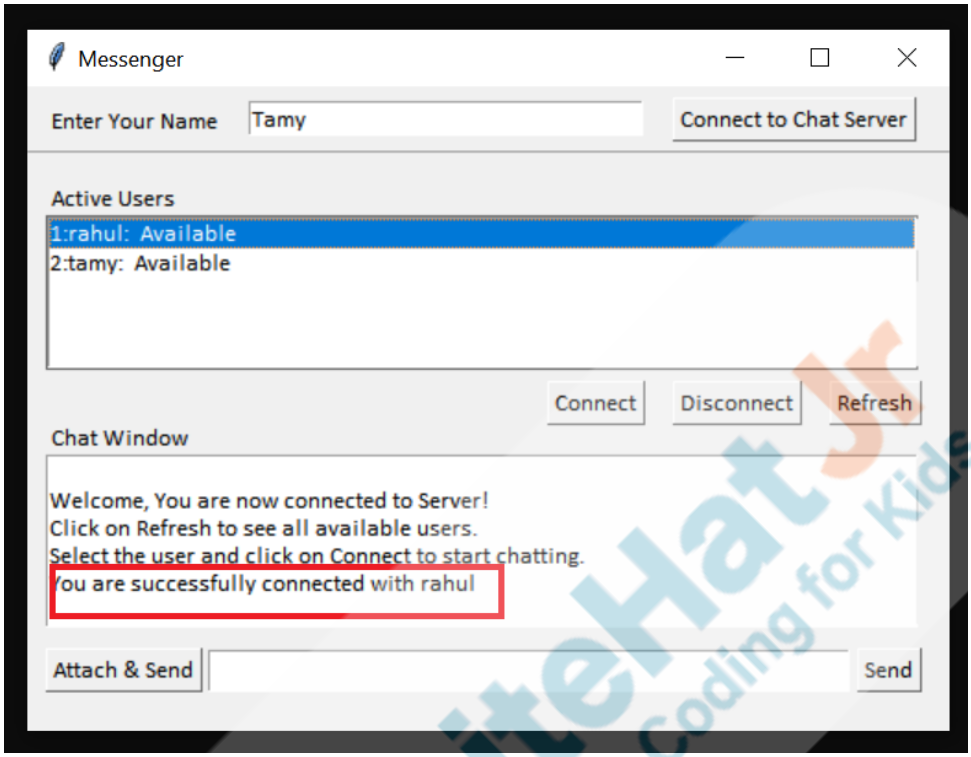
```
                    IP MESSENGER

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

***client.py*** in the terminal/cmd looks like -

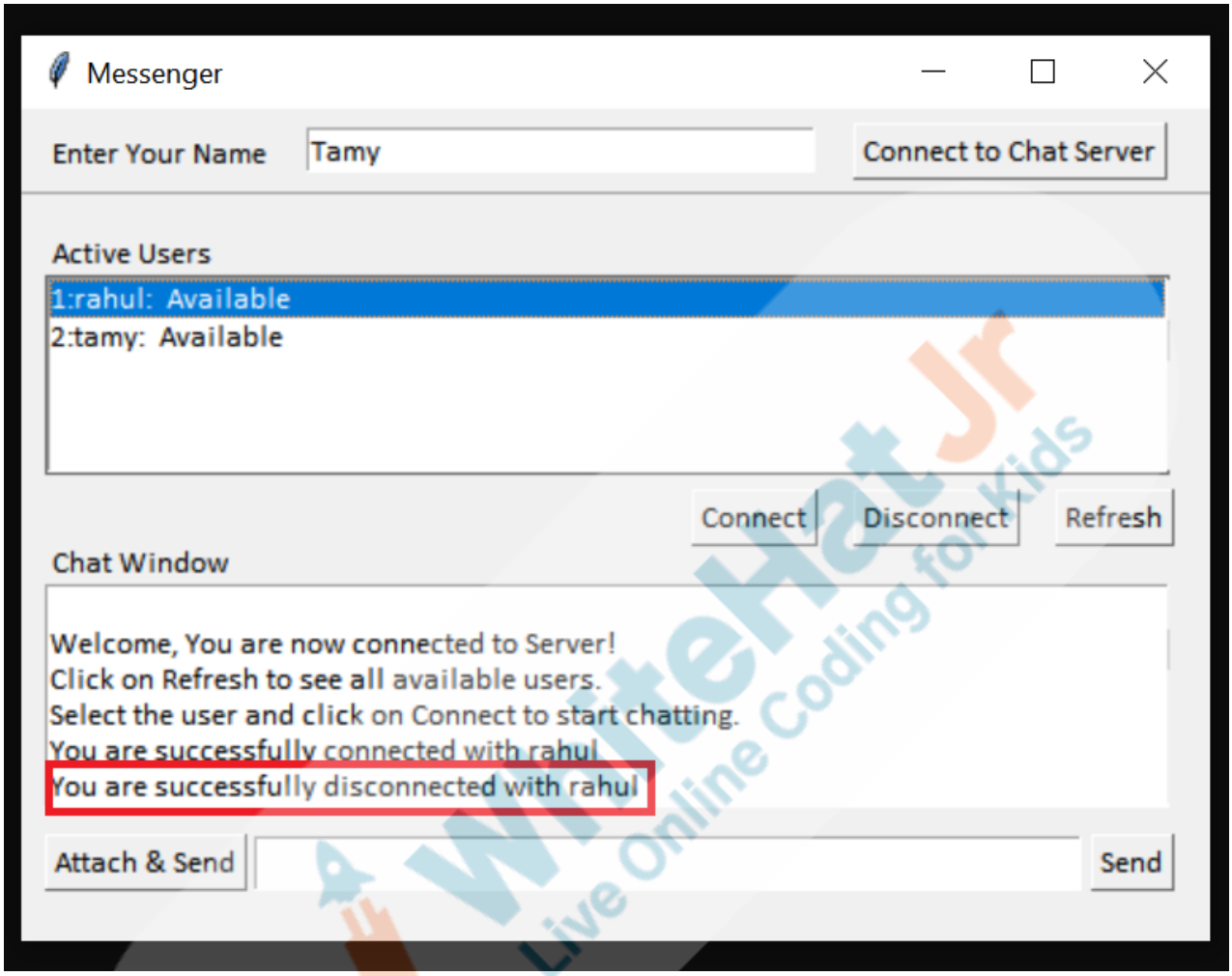


| | |
|---|---|
| *You need to rerun client.py in order to connect to someone*<br>*On clicking refresh button it will display user list, below window will appear like this* | |
|  | |
| *Note :Select the user from list box to connect with other client*<br><br>*On clicking connect button it will display user list, below window will appear like this* | |




Note :Select the user from list box to connect with other client

On clicking Disconnect button it will display user list, below window will appear like this




| | |
|---|---|
| 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 our lower() method? | **A** |

| | |
|---|---|
| A. Convert upper case string to lowercase()<br>B. Convert lower case string to uppercase()<br>C. Convert to numbers<br>D. None of the above | |
| What is the purpose of encoding in sockets?<br><br>A. To convert bytes to strings<br>B. To convert strings to bytes<br>C. Convert string to numbers<br>D. Convert numbers to strings | **A** |
| What is the use of the split method?<br><br>A. Join two strings<br>B. Concatenate two strings<br>C. Divide Two strings<br>D. Split data into smaller Chunks | **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!<br><br>In the next class | *Make sure you have given at least 2 Hats Off during the class for:*<br>  <br>  |

| | |
|---|---|
| | Strong Concentration +10 |
| **Project Discussion**<br><br>**Goal of the Project:**<br><br>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.<br><br>**Story:**<br><br>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**<br>*Encourage the student to write reflection notes in their reflection journal using markdown.*<br><br>Use these as guiding questions:<br>● What happened today? | *The student uses the markdown editor to write her/his reflections in the reflection journal.* |

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

| ACTIVITY LINKS | | |
|---|---|---|
| **Activity Name** | **Description** | **Link** |
| Teacher Activity1 | Boilerplate Code | https://github.com/pro-whitehatjr/PRO-C210-TeacherBoilerPlateCode |
| Teacher Activity 2 | Reference Code | https://github.com/pro-whitehatjr/PRO-C210-RefrenceCode |
| Student Activity 1 | Boilerplate Code | https://github.com/pro-whitehatjr/PRO-C210-StudentBoilerPlateCode |
| Teacher Reference In-Class Quiz | In-Class Quiz | https://s3-whjr-curriculum-uploads.whjr.online/fb99a835-2cd6-47a4-a6e2-ac5a656f1137.pdf |