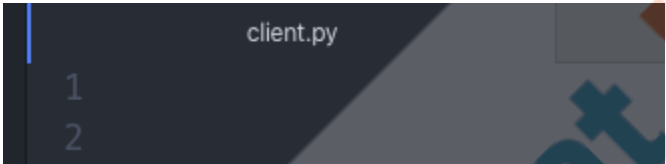


Topic	SOCKETS - CLIENT	
Class Description	Students will learn about Client Socket and will create multi-client chat app with nicknames	
Class	C-200	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> <li>Understand about Client Programming</li> <li>Building Client in Python</li> <li>Application on Socket</li> </ul>	
Resources Required	<ul style="list-style-type: none"> <li>Teacher Resources:               <ul style="list-style-type: none"> <li>Laptop with internet connectivity</li> <li>Earphones with mic</li> <li>Notebook and pen</li> <li>Visual Studio Code</li> </ul> </li> <li>Student Resources:               <ul style="list-style-type: none"> <li>Laptop with internet connectivity</li> <li>Earphones with mic</li> <li>Notebook and pen</li> <li>Visual Studio Code</li> </ul> </li> </ul>	
Class structure	<b>Warm-Up</b> <b>Teacher - led Activity 1</b> <b>Student - led Activity 1</b> <b>Wrap-Up</b>	<b>10 mins</b> <b>10 mins</b> <b>20 mins</b> <b>5 mins</b>
<b>WARM UP SESSION - 10mins</b>		
<b>Teacher Action</b>		<b>Student Action</b>
<i>Hey &lt;student's name&gt;. How are you?            It's great to see you! Are you excited to learn something new today?</i>		<b>ESR:</b> Hi, thanks, Yes I am excited about it!

Q&A Session	
Question	Answer
Which method is used to close the socket?  A. Flush() B. send() C. close() D. accept()	C
Which method of the socket module allows a server socket to accept requests from a client socket?  A. Socket.accept() B. Accept.socket() C. Socket send address() D. Socket. socketaccept()	A
TEACHER-LED ACTIVITY - 10mins	
Teacher Initiates Screen Share	
<p style="text-align: center;"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> <li>• Learning about Sockets and Client Programming</li> <li>• Building Client server with Python</li> </ul>	
Teacher Action	Student Action
In the last class, we learned about sockets in detail and we created a socket server through python programming.  Any doubts from the last session?  <i>Teacher resolves the query of the</i>	<b>ESR:</b> Varied!

<p><i>student (if any)</i></p> <p>Great!</p> <p>In the last class we created only a socket server and in this class we are going to create a client side. This way, our chat app will be functional! Before we begin with the coding, let me ask you a question. Can you explain what the client side is?</p>	<p><b>ESR:</b> Varied!</p>
<p>Client side means your computer or other computers who are requesting the data from the server.</p> <p>Let's understand with this example</p> <p>When a client (your computer) makes a request for any web page (like whitehatjr.com) that information is processed by the server through which we get output which is visible on our computer screen.</p> <p>Now the main part we need to understand is how we can connect to the server. Right?</p> <p>Most of it is similar to how the server was created.</p> <p>Let's learn how we create a client file and how we can connect with the server.</p> <p>The client side script will simply attempt to access the server socket.</p>	

<p><b>Clients</b> are Front-end functions which simply attempt to access the server socket.</p>	
<p>Let's Create Client side</p> <p><i>Teacher opens a new Visual Studio Code and creates a file called <b>client.py</b></i></p>	
	
<p>As discussed in the last session, which library can we use in python for creating sockets?</p> <p>Awesome!</p> <p>Let's import socket Library</p>	<p><b>ESR?</b> We can use the <b>socket</b> library!</p>
<p><i>Teacher opens <a href="#">Teacher Activity 1</a> to refer to the <b>socket</b> library's documentation</i></p> <p>Let's start by importing that -</p>	<p><i>Student opens <a href="#">Student Activity 1</a> to refer to the <b>socket</b> library's documentation</i></p>
<div data-bbox="630 1360 946 1423" data-label="Text"> <pre>import socket</pre> </div> <p>Again, we will now create a TCP socket called <b>client</b> this time. It will follow the Address Family of IPv4, just as how we did in the last class -</p> <div data-bbox="186 1583 1388 1749" data-label="Text"> <pre>import socket  client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)</pre> </div> <p>Here, we are creating a socket with the help of <b>socket.socket()</b> function. With this, our</p>	

socket for the client side is created.

As discussed same in the last class, the socket function takes 2 main arguments -

1. **address family**
2. **socket type**

Now since it is a chat application, we also want our clients to have a nickname, yes? Let's take input from them before we define a client. Our code would look like -

```
import socket

nickname = input("Choose your nickname: ")

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Can you tell me what is the purpose of two main arguments in the socket function.

What is the use of Address family?

What is the purpose of the 2nd Argument SOCK\_STREAM?

#### ESR

Two arguments in socket function:  
Address family & socket type

**address family** is the family of addresses that the socket can communicate with. Classic examples of 2 of the most famous address families are **IPv4** and **IPv6**.

**AF\_INET** represents **IPv4** while **AF\_INET6** represents **IPv6**.

**SOCK\_STREAM** is used to create a **TCP Socket**. **SOCK\_DGRAM** which is used to create a **UDP Socket**.

Great!	
Next, we can define the IP Address and the Port we are using -	
<pre>ip_address = '127.0.0.1' port = 8000</pre> <p>Since our localhost IP Address is <b>127.0.0.1</b>, we will be using that here and we can go with port <b>8000</b>, however this port can be any number. Just make sure that it is not anything lower than <b>1,024</b>, since those are reserved ports.</p>	
<p>So far, everything looks similar to what we have done in the previous class.</p> <p>After defining the IP Address and the Port, we used a <b>bind()</b> function to bind the server socket with them.</p> <p>This time, we are trying to connect with the server that has the IP Address and the Port we defined above.</p> <p>For that, the client socket we created can use another special function offered by socket called <b>connect()</b>.</p> <p><b>connect()</b> method attempts to establish a connection between two sockets. Basically it will help the client application (your computer) to establish a connection to a server.</p>	

```
client.connect((ip_address, port))
```

Here, with the **connect()** function, we are passing the IP Address and the Port to which our server was binded as a tuple, so it will know where it needs to connect.

Now just our showcase or for our information we can use print statement where we will print "connected with the server"

```
print("Connected with the server...")
```

Our code until now looks like this -

```
client.py

import socket

nickname = input("Choose your nickname: ")

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

ip_address = '127.0.0.1'
port = 8000

client.connect((ip_address, port))

print("Connected with the server...")
```

Now just like how our server had to keep listening for client's connection requests, our client will have to keep listening for all the messages that it receives.

They could be from the server, like a **"Welcome to the chatroom"** message or they could be from the client.

How can we continuously listen to all the messages that the client receives?

Excellent! Let's create a function called **receive()** that will receive the messages and add a **while True** loop in inside it then -

**ESR:**

By using a **while True** loop



```
print("Connected with the server...")

def receive():
    while True:
```

Inside this loop, we want to receive the messages from the server. It can be done with the **recv()** function that we learnt in the last class!

```
def receive():
    while True:
        try:
            message = client.recv(2048).decode('utf-8')
        except:
            print("An error occurred!")
            client.close()
            break
```

Here, just to ensure we handle the errors, if any, we are keeping a **try except** statement which, if an error occurs, would let the client know and close the client's socket with the **close()** function. We are also breaking out of the loop in the **except** statement.

Now, in the **try** statement, we are receiving the client's message with the **recv()** function, allowing a maximum of **2048** characters. Just how the messages were **encoded** in the server side with **utf-8**, here, we are **decoding** it with **utf-8**.

Okay so now, we receive the message, what now?

**ESR:**

We will display the message to the client.

Awesome! But there's one more thing to do. We want to add the ability for client's to choose a nickname for themselves. We are also taking input from the user on what their nickname should be at the beginning of our code.

This function is made for **receiving()** messages, however, if you remember at the server side, we used to start a thread that listens to messages from a client after the client makes a connection. If we want to take the nickname in the server, we can do that by sending a special message to the client on which it can let the server know the nickname. Let's do that -

```
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 occurred!")  
            client.close()  
            break
```

Here, you will notice the **if else** statement. Here, the if statement has a condition where it checks if the message is **NICKNAME**. Assuming that this is the message our server will send to the client to know their nickname, the client can send back the nickname with the help of the **send()** function of the socket.

If however, the message is anything **else** than **NICKNAME**, we can simply just print the message.

With this, our **receive()** function is done.

Now, just as we have created a function to **receive()** messages from

the server, we also want to create another **write()** function which can send the server the messages client types!

This function will again run endlessly with the **while True** loop! Let's build it -

```
def write():  
    while True:
```

We start by creating the function to write and add a **while True** loop in it. Inside this loop, we will ask the user to first provide an input. We can do that by writing -

```
def write():  
    while True:  
        message = '{}: {}'.format(nickname, input(''))
```

Here, if you notice, our message is a **string** with “” and we are using **string formatting** in Python with the **nickname** of the client as the value in the first curly braces {}, **input()** from the user in the second curly braces. Whatever the user inputs will become a part of the string. Isn't it interesting?

Next, we simply just send the message with the **send()** function.

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

Alright, we are almost done here! Now we just need to call these functions. We want to run these functions parallelly, which means that the client will receive the messages and write

**ESR:**  
Threads!

back to the server both at the same time, therefore we will be using?

Awesome! Let's do that -

```
import socket
from threading import Thread
```

We will start with importing the **Thread**

Next, we will simply just create the threads and start them with the **start()** function -

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

Now our client's side code is done! We have one thing left to do, and that is to integrate the nickname's logic into the server!

**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 the nickname's logic to the server

**Teacher Action**

**Student Action**

Guide the student to get the boilerplate code from [Student Activity 2](#)

Student clones the code from [Student Activity 2](#)

The first thing that we should do is to create another list of nicknames next to our list of clients in our **server.py** we created in the last class.

*Teacher helps the student in writing the code*

**ESR:**

*Student writes the code*

```
list_of_clients = []  
nicknames = []
```

Here, this list will help us save the **nicknames** of the clients.

Next, inside our **while True** loop, when we set our server to accept connection requests from clients with the **accept()** function, we can send the **NICKNAME** message to the client right at that moment. This way, we will get the nickname as soon as the client connects with the server.

Let's write the code for that -

*Teacher helps the student in writing the code*

*Student writes the code*

The code changes from this -

```
while True:
    conn, addr = server.accept()
    list_of_clients.append(conn)
    print (addr[0] + " connected")
    new_thread = Thread(target= clientthread,args=(conn,addr))
    new_thread.start()
```

To this -

```
while True:
    conn, addr = server.accept()
    conn.send('NICKNAME'.encode('utf-8'))
    nickname = conn.recv(2048).decode('utf-8')
    list_of_clients.append(conn)
    nicknames.append(nickname)
    message = "{} joined!".format(nickname)
    print(message)
    broadcast(message, conn)
    new_thread = Thread(target= clientthread,args=(conn, nickname))
    new_thread.start()
```

Here, you will notice that we added a few extra lines of code.

The first change you'll notice is that right after making our server accept connection requests from the clients with the **accept()** function, we are using the **send()** function to send the message **NICKNAME** to the client, and then we are receiving the nickname with the **recv()** method.

Next, we are appending the nickname to the **nicknames** list we created earlier.

Instead of printing the IP address of the client, we are not printing the nickname.

We are also broadcasting the **"{nickname} Joined!"** message to all the other clients,

with the ***broadcast()*** function we created earlier.

Lastly, we have added a new argument ***nickname*** to our ***clientthread*** function while creating the thread. Also, since we are now using the nickname, we don't need to pass the ip address now.

Just as we added a new argument ***nickname*** and removed ***addr*** from the ***clientthread()*** function, the same changes will reflect in the function declaration too -

```
def clientthread(conn, nickname):  
    conn.send("Welcome to this chatroom!".encode('utf-8'))  
    while True:
```

Also, inside this function, instead of printing the message with the IP Address as we are doing, we can simply just print the message now.

Our code for ***clientthread()*** becomes from this -

```
def clientthread(conn, addr):  
    conn.send("Welcome to this chatroom!".encode('utf-8'))  
    while True:  
        try:  
            message = conn.recv(2048).decode('utf-8')  
            if message:  
                print("<" + addr[0] + "> " + message)  
  
                message_to_send = "<" + addr[0] + "> " + message  
                broadcast(message_to_send, conn)  
            else:  
                remove(conn)  
        except:  
            continue
```

To this -

```
def clientthread(conn, nickname):  
    conn.send("Welcome to this chatroom!".encode('utf-8'))  
    while True:  
        try:  
            message = conn.recv(2048).decode('utf-8')  
            if message:  
                print(message)  
                broadcast(message, conn)  
            else:  
                remove(conn)  
                remove_nickname(nickname)  
        except:  
            continue
```

Now, one last thing that you may notice is that we have a new function called **`remove_nickname()`**. This is because we now want to remove the nickname of the client too, when we are removing the client socket itself.

Let's create the function for it -

```
def remove_nickname(nickname):  
    if nickname in nicknames:  
        nicknames.remove(nickname)
```

Now to test this, we are going to need 3 terminals/command prompts at least.

*Teacher helps the student in opening 3 command prompts/terminals*

Run

**`python3.8 server.py`**

*Student opens 3 command prompts/terminals*



In 1 window and run

**`python3.8 client.py`** in 2 remaining windows

*Teacher helps the student in running the commands*

*Student runs the commands*

Our Server would look like -

```
Server has started...
John joined!
Bob joined!
```

Our Clients would look like -

```
Choose your nickname: John
Connected with the server...
Welcome to this chatroom!
Bob joined!
```

And

```
Choose your nickname: Bob
Connected with the server...
Welcome to this chatroom!
```

Now, if you type “**Client 1**” in one client window and press enter, type “**Client 2**” in the

second client window and press enter, the output would look like -

***Client 1 Window -***

```
Choose your nickname: John
Connected with the server...
Welcome to this chatroom!
Bob joined!
Client 1
Bob: Client 2
```

***Client 2 Window -***

```
Choose your nickname: Bob
Connected with the server...
Welcome to this chatroom!
John: Client 1
Client 2
```

***And the Server Window -***

```
Server has started...
John joined!
Bob joined!
John: Client 1
Bob: Client 2
```




In the next class, we will add a GUI to this application.

**Teacher Guides Student to Stop Screen Share**

**WRAP UP SESSION - 5 Mins**

**Quiz time - Click on in-class quiz**

Question	Answer
<p>The client in socket programming must know which information?</p> <p>A. IP Address B. Port C. IP Address &amp; Port D. None of the above</p>	<b>C</b>
<p>Which method is used for taking input from the client side?</p> <p>A. sys.stdin() B. sys.stdout() C. input() D. system.stdin()</p>	<b>A</b>
Which method is used to flush the	

buffer?  A. sys.stdout.flush() B. system.flush() C. flush() D. flush.system()	<b>A</b>
<b>End the quiz panel</b>	
<b>FEEDBACK</b> <ul style="list-style-type: none"> <li>• Appreciate the students for their efforts in the class.</li> <li>• Ask the student to make notes for the reflection journal along with the code they wrote in today's class.</li> </ul>	
Teacher Action	Student Action
You get Hats off for your excellent work!   In the next class	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div>           Creatively Solved Activities  +10         </div> <div>           Great Question  +10         </div> <div>           Strong Concentration  +10         </div>
<b>Project Discussion</b>	
Teacher Clicks	<div>✕ End Class</div>
<b>ADDITIONAL ACTIVITIES</b>	
<b>Additional Activities</b>	<i>The student uses the markdown editor to write her/his reflections in the reflection journal.</i>

*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?

ACTIVITY LINKS		
Activity Name	Description	Link
Teacher Activity 1	Socket Library	<a href="https://docs.python.org/3/library/socket.html">https://docs.python.org/3/library/socket.html</a>
Teacher Activity 2	Reference Code	<a href="https://github.com/pro-whitehatjr/C200_TeacherReference">https://github.com/pro-whitehatjr/C200_TeacherReference</a>
Student Activity 1	Socket Library	<a href="https://docs.python.org/3/library/socket.html">https://docs.python.org/3/library/socket.html</a>
Student Activity 2	Boilerplate Code	<a href="https://github.com/pro-whitehatjr/C200_SA_BoilerPlate">https://github.com/pro-whitehatjr/C200_SA_BoilerPlate</a>