

## SOCKETS - SERVER



### What is our GOAL for this CLASS?

In this class, we have learned about sockets and threads. We also learnt about the **socket** module in Python and did **socket programming** to start building a terminal based chat room application.

### What did we ACHIEVE in the class TODAY?

- Understanding and learning Sockets
- Socket Module in Python
- Threads and Multithreading
- Socket Programming

### Which CONCEPTS/ CODING BLOCKS did we cover today?

- Python
- Sockets
- Threads
- TCP/UDP

### How did we DO the activities?

In this class, we learnt about sockets and understood that -

- Sockets allow communication between two different applications
- They are used in a **client-server application framework**

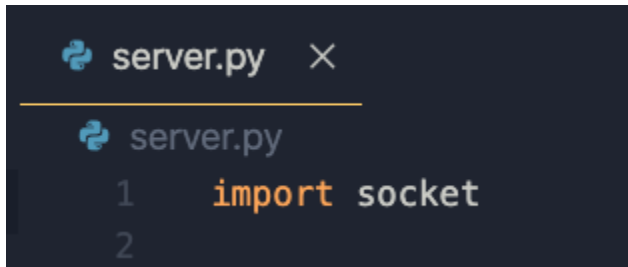
**Servers** are backend applications that perform some functions on a request from the **Client**.

We also understood the differences between TCP and UDP -

TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Tries to resend the packets that are lost	Doesn't resend the packets that are lost
Adds a sequence number to the packets and reorders them to ensure the packets do not arrive in wrong order	Packets can arrive in any order.
Slower, as it manages everything	Faster due to lack of features
Heavy to use as OS keeps track of ongoing communication sessions between clients and servers	Lighter to use on the machine
TCP is used by - <ul style="list-style-type: none"><li>• HTTP</li><li>• HTTPS</li><li>• SMTP</li><li>• FTP</li><li>• etc.</li></ul>	UDP is used by - <ul style="list-style-type: none"><li>• DNS</li><li>• DHCP</li><li>• etc.</li></ul>

**Coding Activity -**

1. Create a new file - **server.py**
2. Import **socket** into it -



```
server.py X
server.py
1  import socket
2
```

3. Create a simple socket where -
  - a. Address Family is **AF\_INET (IPv4)**
  - b. Socket Type is **SOCK\_STREAM (TCP)**

```
import socket

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

4. Define IP Address to use and a Port Number for the server

```
ip_address = '127.0.0.1'
port = 8000
```

5. Bind the server with IP and Port and then listen -

```
server.bind((ip_address, port))
server.listen()
```

6. Create a list to store all the clients that will connect -

```
clients = []
```

Our code Until Now -

```
import socket

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

ip_address = '127.0.0.1'
port = 8000

server.bind((ip_address, port))
server.listen()

clients = []

print("Server is running...")
```

7. To make the server run indefinitely, make a **while True** loop -

```
while True:
    conn, addr = server.accept()
    list_of_clients.append(conn)
    print (addr[0] + " connected")
```

- **accept()** is a special method of the server socket that we created
- It returns the socket object of the client that wants to connect and their IP Address

8. Import Thread from threading -

```
from threading import Thread
```

9. Create new thread for all client connections inside the while loop -

```
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()
```

10. Create the **clientthread()** function that actively listens to messages received from a client and broadcasts them -

```
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
```

11. Create the **broadcast()** function that sends the messages to the clients -

```
def broadcast(message, connection):  
    for clients in list_of_clients:  
        if clients!=connection:  
            try:  
                clients.send(message.encode('utf-8'))  
            except:  
                remove(clients)
```

12. Create the **remove()** function to remove a client from the client list -

```
def remove(connection):  
    if connection in list_of_clients:  
        list_of_clients.remove(connection)
```

### What's NEXT?

In the next class, we will continue working on our chat application and build the client side's code.

### Expand Your Knowledge:

Explore the socket's documentation [here](#).