PD LAB
ASSIGNMENT - 3

Name: Raunak Thanawala
Registration Number: 231070051
Branch: Computer Engineering
Batch: 3

Aim:-
To create a messaging application in python using socket programming and tkinter.

Theory:-
Socket programming is a way to enable communication between two terminals over a network.
Sockets provide a way for software applications to send and receive data, allowing for network-based communication.

It is a fundamental technology for creating networked applications like web servers, chat clients, and multiplayer games.
A socket is an endpoint for sending or receiving data across a computer network.

Types of Sockets:

Stream Sockets (TCP):
Provide a reliable, two-way, connection-based byte stream.
TCP (Transmission Control Protocol) ensures that data is delivered accurately and in order.

Datagram Sockets (UDP):
Provide a connectionless, unreliable messaging service.
UDP (User Datagram Protocol) is faster and more efficient for applications that do not require guaranteed delivery of data.

Client-Server Model: Socket programming often follows a client-server architecture, where the server waits for incoming connections, and the client initiates a connection to the server.

Steps for Socket Programming

1. Create a Socket:
   A socket is created using the socket() function, specifying the address family (such as IPv4 or IPv6) and the socket type (such as TCP or UDP).

2. Bind the Socket (For Servers):
   Bind the socket to an IP address and port number using the bind() method, which specifies the address and port to listen for incoming connections.

3. Listen for Connections (For Client): For a server, the socket needs to listen for incoming connections using the listen() method, which allows the server to accept incoming requests.

4. Accept Connections (For Server): The server uses the accept() method to accept a connection from a client. This method returns a

new socket object representing the connection and the address of the client.

5. Connect to the Server (For Client): For clients, connect to the server using the connect() method, which establishes a connection to the server's socket.

6. Send and Receive Data : Data is transmitted between the client and server using the send() and recv() methods for TCP sockets, or sendto() and recvfrom() for UDP sockets.

7. Close the Socket (For Server): Once the communication is finished, close the socket using the close() method to free up resources.

Code and Output:

1. Server Code:

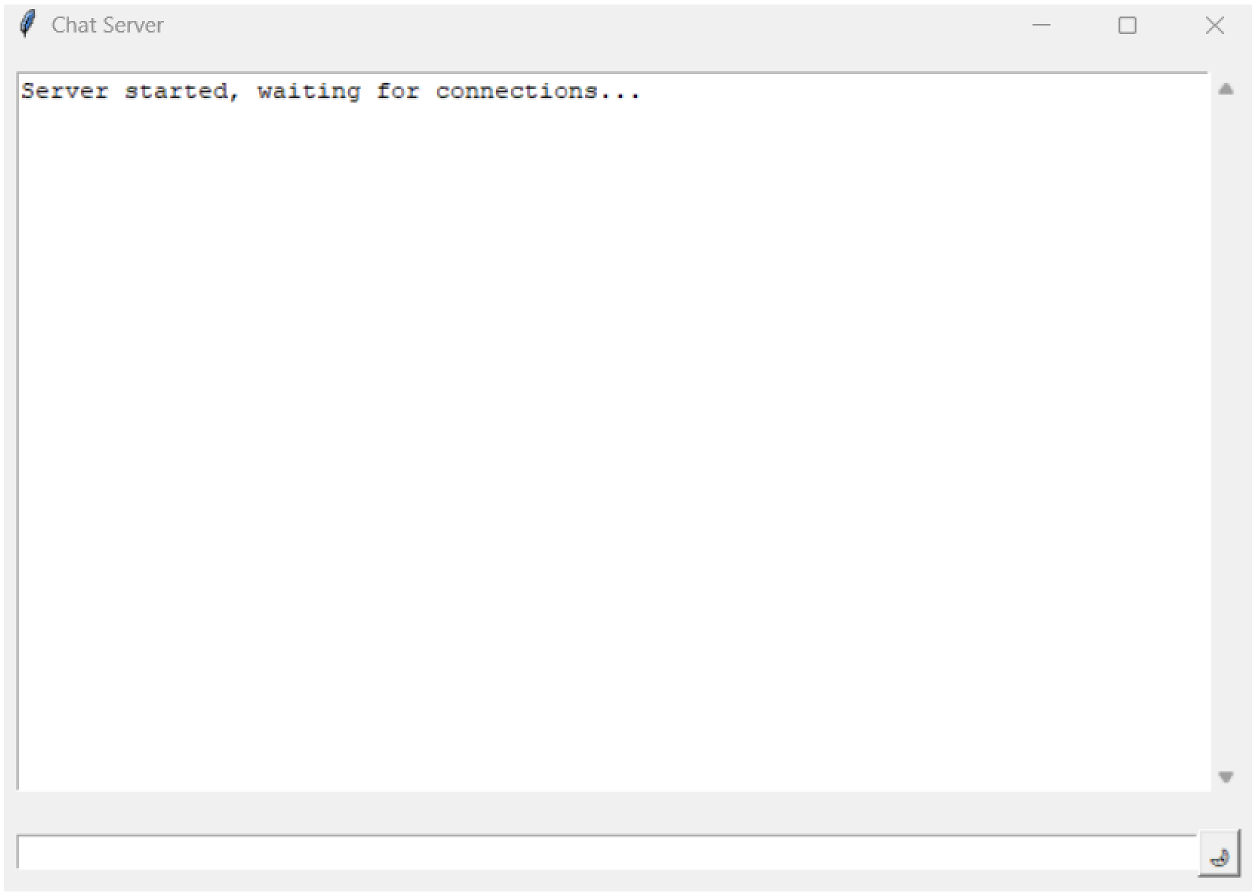
```
import socket
import threading
import tkinter as tk
from tkinter import scrolledtext

HOST = '127.0.0.1'
PORT = 12340

class ChatServer:
    def __init__(self, root):
        self.root = root
        self.root.title("Chat Server")
        self.dark_mode = False

        # Create GUI components
        self.chat_area = scrolledtext.ScrolledText(root, state='disabled')
        self.chat_area.pack(padx=10, pady=10)

        self.message_frame = tk.Frame(root)
        self.message_frame.pack(padx=10, pady=10, fill=tk.X)

        self.message_entry = tk.Entry(self.message_frame)
        self.message_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)
        self.message_entry.bind("<Return>", self.send_message)

        self.toggle_button = tk.Button(self.message_frame, text="🌙",
command=self.toggle_mode)
        self.toggle_button.pack(side=tk.RIGHT)

        # Initialize server socket and client list
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.clients = {}  # Dictionary to hold clients and their usernames

        # Start the server
        try:
            self.server_socket.bind((HOST, PORT))
            self.server_socket.listen()
            self.update_chat_area("Server started, waiting for connections...")
        except socket.error as e:
            self.update_chat_area(f"Socket error: {e}")
            self.root.quit()
            return

        # Start thread to accept connections
        threading.Thread(target=self.accept_connections, daemon=True).start()

    def accept_connections(self):
        """Accept new client connections."""
        while True:
```

```python
            try:
                client_socket, client_address = self.server_socket.accept()
                threading.Thread(target=self.handle_client,
args=(client_socket,), daemon=True).start()
            except socket.error as e:
                self.update_chat_area(f"Socket error while accepting connections:
{e}")

    def handle_client(self, client_socket):
        """Handle incoming messages from a client."""
        try:
            username = client_socket.recv(1024).decode('utf-8')
            self.clients[client_socket] = username
            self.update_chat_area(f"Connection from {username}
({client_socket.getpeername()})")
        except socket.error as e:
            self.update_chat_area(f"Socket error while receiving username: {e}")
            client_socket.close()
            return

        while True:
            try:
                message = client_socket.recv(1024).decode('utf-8')
                if not message:  # Client has disconnected
                    break
                self.broadcast(f"{username}: {message}", client_socket)
            except socket.error as e:
                self.update_chat_area(f"Socket error while handling client
{username}: {e}")
                break

        # Cleanup client connection
        self.remove_client(client_socket)

    def broadcast(self, message, source_socket):
        """Broadcast a message to all clients except the source socket."""
        self.update_chat_area(message)

        # Broadcast the message to all clients
        for client in list(self.clients):
            if client != source_socket:
                try:
                    client.send(message.encode('utf-8'))
                except socket.error:
                    self.remove_client(client)

    def send_message(self, event=None):
        """Send a server message to all clients."""
        message = self.message_entry.get()
        if message:
            self.broadcast(f"Server: {message}", None)
            self.message_entry.delete(0, tk.END)

    def update_chat_area(self, message):
        """Update the chat area with a new message."""
        self.chat_area.configure(state='normal')
        self.chat_area.insert(tk.END, message + '\n')
        self.chat_area.configure(state='disabled')
        self.chat_area.yview(tk.END)

    def toggle_mode(self):
        """Toggle between light and dark mode."""
        if self.dark_mode:
            self.root.configure(bg="white")
            self.chat_area.configure(bg="white", fg="black")
            self.message_entry.configure(bg="white", fg="black")
            self.toggle_button.configure(bg="lightgrey", fg="black")
            self.dark_mode = False
        else:
            self.root.configure(bg="black")
            self.chat_area.configure(bg="black", fg="white")
            self.message_entry.configure(bg="black", fg="white")
            self.toggle_button.configure(bg="darkgrey", fg="white")
            self.dark_mode = True

    def remove_client(self, client_socket):
        """Remove a client from the list and close its connection."""
        if client_socket in self.clients:
            username = self.clients.pop(client_socket)
            client_socket.close()
            self.update_chat_area(f"{username} ({client_socket.getpeername()})
disconnected.")

if __name__ == "__main__":
    root = tk.Tk()
    server = ChatServer(root)
    root.mainloop()
```

2.Client Code:

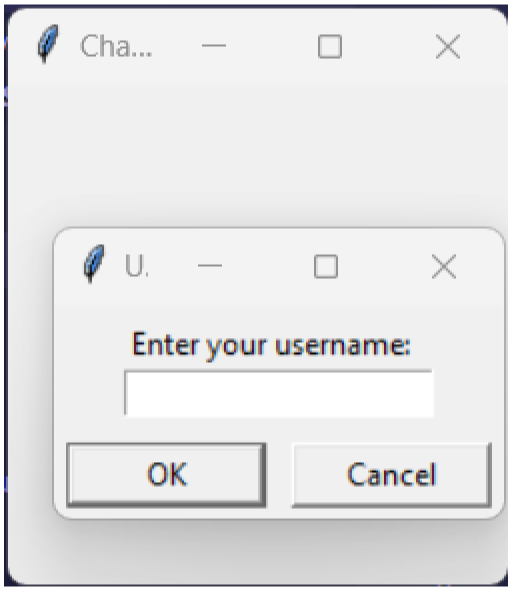
```
import tkinter as tk
from tkinter import simpledialog, scrolledtext
import socket
import threading

HOST = '127.0.0.1'
PORT = 12340

class ChatClient:
    def __init__(self, root):
        self.root = root
        self.root.title("Chat Client")
        self.dark_mode = False

        # Prompt for username
        self.username = simpledialog.askstring("Username", "Enter your
username:", parent=root)
        if not self.username:
            self.root.quit()
            return

        # Create GUI components
        self.chat_area = scrolledtext.ScrolledText(root, state='disabled')
        self.chat_area.pack(padx=10, pady=10)
        self.message_frame = tk.Frame(root)
        self.message_frame.pack(padx=10, pady=10, fill=tk.X)

        self.message_entry = tk.Entry(self.message_frame)
        self.message_entry.pack(side=tk.LEFT, fill=tk.X, expand=True)
        self.message_entry.bind("<Return>", self.send_message)

        self.toggle_button = tk.Button(self.message_frame, text="🌙",
command=self.toggle_mode)
        self.toggle_button.pack(side=tk.RIGHT)

        # Initialize and connect socket
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.socket.connect((HOST, PORT))
            self.socket.send(self.username.encode('utf-8'))  # Send username to
server
        except socket.error as e:
            self.update_chat_area(f"Connection error: {e}")
            self.root.quit()
            return

        # Start the receive thread
        threading.Thread(target=self.receive_messages, daemon=True).start()

    def send_message(self, event=None):
        """Send a message to the server."""
        message = self.message_entry.get()
        if message:
            try:
                self.socket.send(message.encode('utf-8'))
                self.message_entry.delete(0, tk.END)
                self.update_chat_area(f"You: {message}")
            except socket.error:
                self.update_chat_area("Failed to send message. Disconnected from
server.")

    def receive_messages(self):
        """Receive messages from the server."""
        while True:
            try:
                message = self.socket.recv(1024).decode('utf-8')
                if message:
                    self.update_chat_area(message)
            except socket.error:
                self.update_chat_area("Connection closed by the server.")
                break

    def update_chat_area(self, message):
        """Update the chat area with a new message."""
        self.chat_area.configure(state='normal')
        self.chat_area.insert(tk.END, message + '\n')
        self.chat_area.configure(state='disabled')
        self.chat_area.yview(tk.END)

    def toggle_mode(self):
        """Toggle between light and dark mode."""
        if self.dark_mode:
            self.root.configure(bg="white")
            self.chat_area.configure(bg="white", fg="black")
            self.message_entry.configure(bg="white", fg="black")
            self.toggle_button.configure(bg="lightgrey", fg="black")
            self.dark_mode = False
        else:
            self.root.configure(bg="black")
            self.chat_area.configure(bg="black", fg="white")
            self.message_entry.configure(bg="black", fg="white")
            self.toggle_button.configure(bg="darkgrey", fg="white")
            self.dark_mode = True

if __name__ == "__main__":
    root = tk.Tk()
    client = ChatClient(root)
    root.mainloop()
```

OUTPUT:

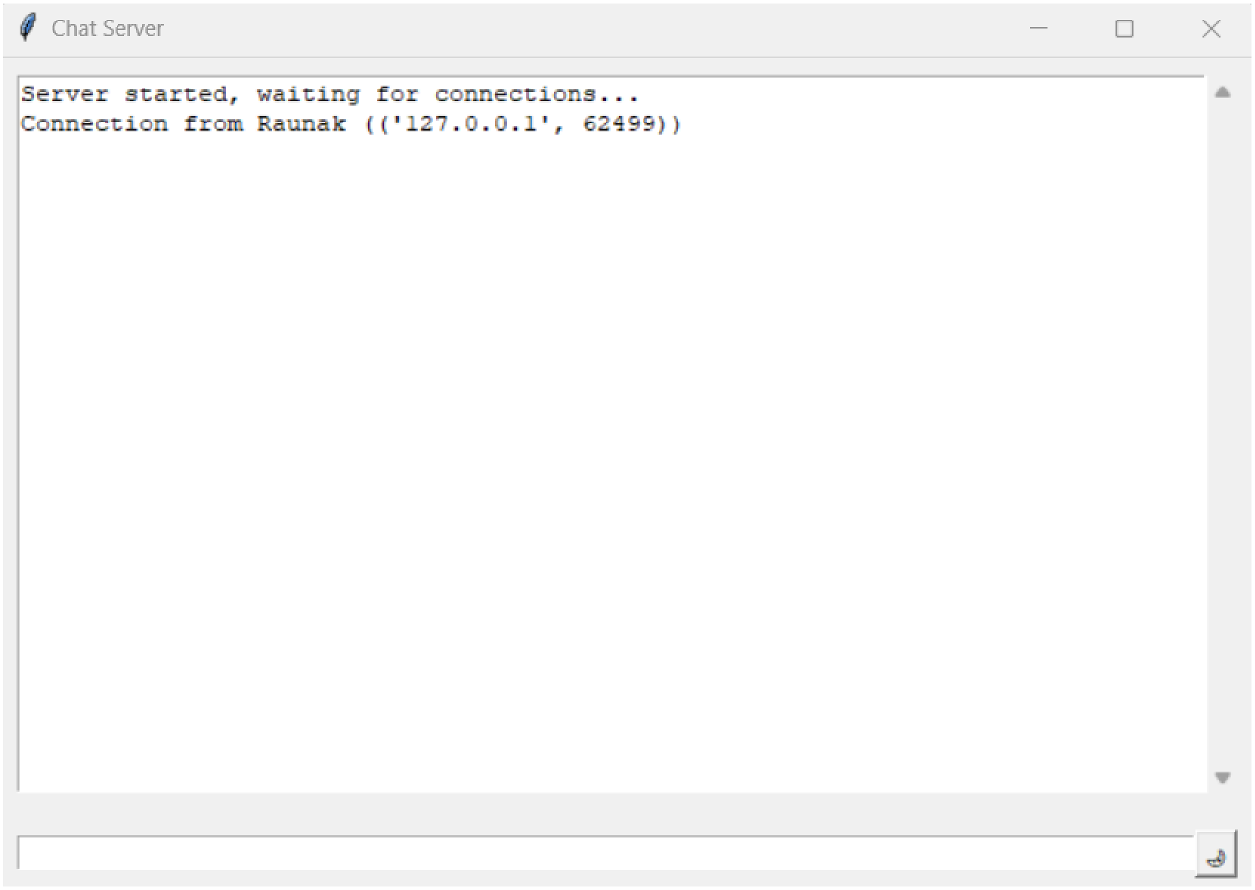


```
Server started, waiting for connections...
```

Initial Output for Server

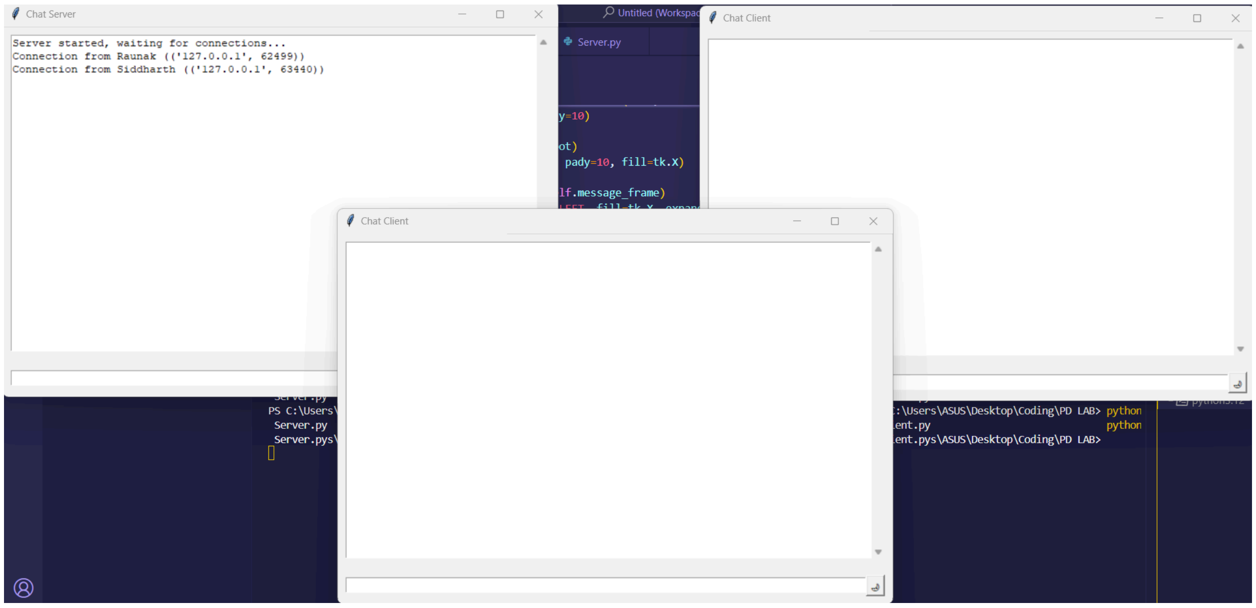


Initial Output for Client



```
Server started, waiting for connections...
Connection from Raunak (('127.0.0.1', 62499))
```

After Entering your name and clicking OK



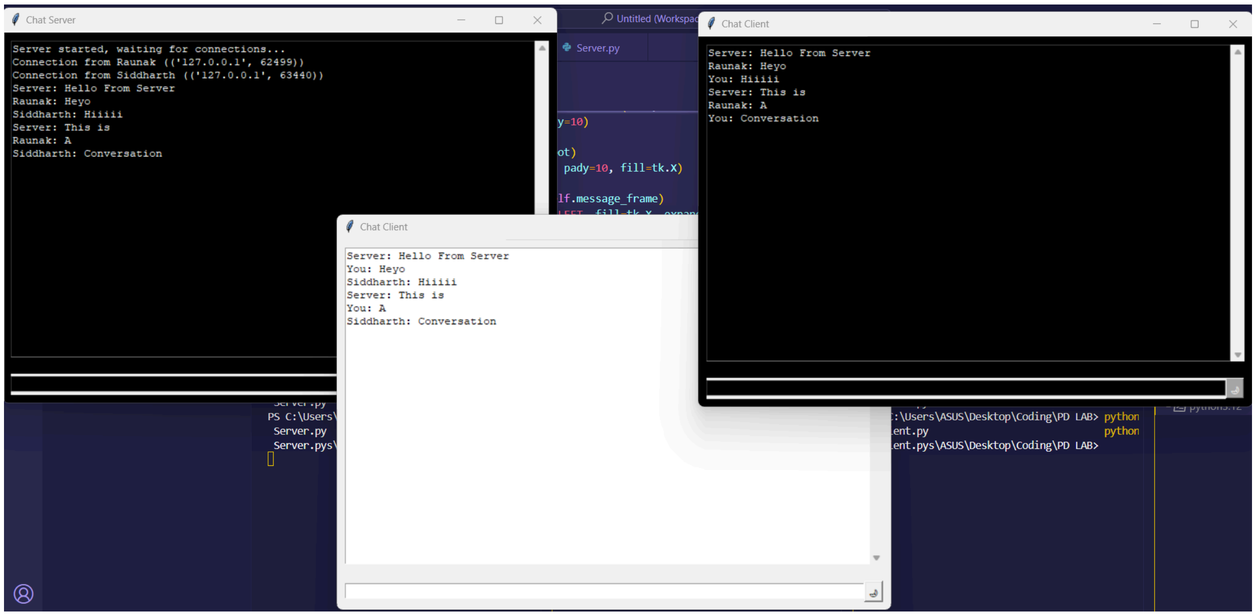
```
Server started, waiting for connections...
Connection from Raunak (('127.0.0.1', 62499))
Connection from Siddharth (('127.0.0.1', 63440))
```

```
y=10)
ot)
 pady=10, fill=tk.X)
lf.message_frame)
```

```
PS C:\Users\
 Server.py
 Server.pys\
```

```
:\Users\ASUS\Desktop\Coding\PD LAB> python
ent.py                               python
ent.pys\ASUS\Desktop\Coding\PD LAB>
```

With Multiple Users



```
Server started, waiting for connections...
Connection from Raunak (('127.0.0.1', 62499))
Connection from Siddharth (('127.0.0.1', 63440))
Server: Hello From Server
Raunak: Heyo
Siddharth: Hiiiii
Server: This is
Raunak: A
Siddharth: Conversation
```

```
Server: Hello From Server
You: Heyo
Siddharth: Hiiiii
Server: This is
You: A
Siddharth: Conversation
```

```
Server: Hello From Server
Raunak: Heyo
You: Hiiiii
Server: This is
Raunak: A
You: Conversation
```

```
PS C:\Users\
 Server.py
 Server.pys\
```

```
:\Users\ASUS\Desktop\Coding\PD LAB> python
ent.py                               python
ent.pys\ASUS\Desktop\Coding\PD LAB>
```

A Conversation between the Server and 2 Clients where the server and one of the clients is using dark mode

Conclusion:

Thus we have written a program to write a messaging application implementing socket programming and using tkinter to create an interface for the server and client to chat.