**Module 17: Network Programming**

1. **Introduction**

Network programming refers to writing programs that **enable communication between two or more devices** (computers) connected via a network (like LAN, WAN, or the Internet).

It allows applications to:

* Exchange data (text, files, messages)
* Access remote resources
* Build client–server models (e.g., web browsers & web servers)

**Key Concepts:**

* **Client:** Sends a request for data or service.
* **Server:** Waits for client requests and sends responses.
* **Socket:** The endpoint for sending/receiving data across the network.
* **Protocol:** The rule set that defines communication (e.g., TCP/IP, UDP).

In Python, network programming is done mainly using the **socket module**.

**2. A Daytime Server**

A *Daytime Server* is a simple example of a server that sends the **current date and time** to the client whenever the client connects.

**⚙️ How it Works:**

1. Server waits for connections on a specific **port number**.
2. When a client connects, the server sends back the **current date and time** as a string.
3. Client prints this data and closes the connection.

**3. Clients and Servers**

| **Role** | **Description** | **Example** |
| --- | --- | --- |
| **Client** | Requests a service or data from a server | Web Browser |
| **Server** | Provides a service or data | Web Server |

Both use **sockets** for communication:

* Server: socket(), bind(), listen(), accept()
* Client: socket(), connect()
* Both: send(), recv()

1. **The Server Program (Python Example)**

# daytime\_server.py

import socket

from datetime import datetime

# Create socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Bind socket to host and port

server\_socket.bind(('127.0.0.1', 5000))

# Start listening for clients

server\_socket.listen(1)

print("Daytime Server is running... Waiting for connections.")

while True:

    client, address = server\_socket.accept()

    print("Connected by", address)

    # Get current date and time

    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Send to client

    client.send(now.encode())

        client.close()

1. **The Client Program (Python Example)**

# daytime\_client.py

import socket

# Define server details

HOST = '127.0.0.1'   # Localhost

PORT = 5000          # Must match server port

# Create socket

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to server

client\_socket.connect((HOST, PORT))

# Receive data (up to 1024 bytes)

data = client\_socket.recv(1024)

print("Server says:", data.decode())

# Close connection

client\_socket.close()

**Steps to Run:**

1. Save both files (daytime\_server.py and daytime\_client.py).
2. Run the server first:

python daytime\_server.py

Daytime Server is running... Waiting for connections.

This message show in terminal of vs code

**Using CMD**

1. Keep your **server** running in VS Code terminal.
2. Open **Windows CMD** (press Win + R, type cmd, Enter).
3. Run your client from there:

python d:\ACTE\1-Python-Tulsidash\daytime\_client.py

Connecting to server...

Connected successfully!

Server says: 2025-10-08 14:12:45

The client will display the server’s date and time in cmd.

Real-Life Examples **Mini Project: Simple Chat Application (Server + Client)**

**🎯 Goal**

Create a **Chat App** where:

* The **Server** runs first and waits for messages.
* The **Client** connects and both can **send & receive messages** back and forth.

🧩 Step 1: Chat Server (chat\_server.py)

import socket

HOST = '0.0.0.0'   # Accepts connections from all local addresses

PORT = 5050        # Use a safe port

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

print("💬 Chat Server is running... Waiting for connection...")

client\_socket, addr = server\_socket.accept()

print("✅ Connected to:", addr)

while True:

    # Receive message from client

    msg = client\_socket.recv(1024).decode()

    if msg.lower() == 'bye':

        print("❌ Client ended the chat.")

        break

    print(f"Client: {msg}")

    # Send reply

    reply = input("You: ")

    client\_socket.send(reply.encode())

    if reply.lower() == 'bye':

        print("✅ Chat closed by server.")

        break

client\_socket.close()

server\_socket.close()

🧩 Step 2: Chat Client (chat\_client.py)

import socket

HOST = '127.0.0.1'  # Same system

PORT = 5050

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print("✅ Connected to Chat Server. Type 'bye' to exit.")

while True:

    # Send message to server

    msg = input("You: ")

    client\_socket.send(msg.encode())

    if msg.lower() == 'bye':

        print("✅ Chat closed by client.")

        break

    # Receive reply

    reply = client\_socket.recv(1024).decode()

    print(f"Server: {reply}")

    if reply.lower() == 'bye':

        print("❌ Server ended the chat.")

        break

client\_socket.close()

**⚙️ Step-by-Step Setup**

**Step 1: Run Server in VS Code Terminal**

1. Open your project folder in VS Code
2. Open a terminal (Ctrl + ~)
3. Run your server code:
4. python chat\_server.py

You should see:

💬 Chat Server is running... Waiting for connection...

🔸 **Keep this VS Code terminal open** — do not stop or close it.

**Step 2: Run Client in CMD**

1. Open a new CMD window

D:

cd \ACTE\1-Python-Tulsidash

Now your prompt should look like this:

D:\ACTE\1-Python-Tulsidash>

Run:

python chat\_client.py

You’ll see:

✅ Connected to Chat Server. Type 'bye' to exit.

Now you can chat between both windows 🎯

You: Hello

Server: Hi there!

Real-Life Examples **Project: Simple Chat Application (Server + Client) with GUI**

🧩 Step 1: Chat Server (chat\_server\_gui.py)

import socket

import threading

import tkinter as tk

from tkinter import scrolledtext

HOST = '0.0.0.0'

PORT = 5050

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

# --- GUI Setup ---

window = tk.Tk()

window.title("💬 Chat Server")

chat\_area = scrolledtext.ScrolledText(window, width=50, height=20, state='disabled')

chat\_area.pack(padx=10, pady=10)

entry = tk.Entry(window, width=40)

entry.pack(side=tk.LEFT, padx=10)

send\_btn = tk.Button(window, text="Send")

send\_btn.pack(side=tk.LEFT)

client\_socket = None  # placeholder

def log\_message(message):

    chat\_area.config(state='normal')

    chat\_area.insert(tk.END, message + "\n")

    chat\_area.config(state='disabled')

    chat\_area.yview(tk.END)

def handle\_client():

    global client\_socket

    client\_socket, addr = server\_socket.accept()

    log\_message(f"✅ Connected to {addr}")

    while True:

        msg = client\_socket.recv(1024).decode()

        if not msg or msg.lower() == 'bye':

            log\_message("❌ Client disconnected.")

            break

        log\_message(f"Client: {msg}")

def send\_message():

    global client\_socket

    if client\_socket:

        msg = entry.get()

        entry.delete(0, tk.END)

        client\_socket.send(msg.encode())

        log\_message(f"You: {msg}")

send\_btn.config(command=send\_message)

# Start listening thread immediately so GUI loads first

threading.Thread(target=handle\_client, daemon=True).start()

log\_message("💬 Waiting for client connection...")

window.mainloop()

🧩 Step 2: Chat Client (chat\_client\_gui.py)

import socket

import threading

import tkinter as tk

from tkinter import scrolledtext

HOST = '127.0.0.1'

PORT = 5050

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

# --- GUI Setup ---

window = tk.Tk()

window.title("💬 Chat Client")

chat\_area = scrolledtext.ScrolledText(window, width=50, height=20, state='disabled')

chat\_area.pack(padx=10, pady=10)

entry = tk.Entry(window, width=40)

entry.pack(side=tk.LEFT, padx=10)

send\_btn = tk.Button(window, text="Send")

send\_btn.pack(side=tk.LEFT)

def log\_message(message):

    chat\_area.config(state='normal')

    chat\_area.insert(tk.END, message + "\n")

    chat\_area.config(state='disabled')

    chat\_area.yview(tk.END)

def receive\_messages():

    while True:

        msg = client\_socket.recv(1024).decode()

        if not msg or msg.lower() == 'bye':

            log\_message("❌ Server disconnected.")

            break

        log\_message(f"Server: {msg}")

def send\_message():

    msg = entry.get()

    entry.delete(0, tk.END)

    client\_socket.send(msg.encode())

    log\_message(f"You: {msg}")

send\_btn.config(command=send\_message)

thread = threading.Thread(target=receive\_messages)

thread.start()

window.mainloop()

**⚙️ Step-by-Step Setup**

**Step 1: Run Server in VS Code Terminal**

1. Open your project folder in VS Code
2. Open a terminal (Ctrl + ~)
3. Run your server code:
4. python chat\_server.py

You should see:

💬 Chat Server is running... Waiting for connection...

🔸 **Keep this VS Code terminal open** — do not stop or close it.

**Step 2: Run Client in CMD**

1. Open a new CMD window

D:

cd \ACTE\1-Python-Tulsidash

Now your prompt should look like this:

D:\ACTE\1-Python-Tulsidash>

Run:

python chat\_client\_gui.py

You’ll see:

✅ Connected to Chat Server. Type 'bye' to exit.

Now you can chat between both windows 🎯

You: Hello

Server: Hi there!