Implementing UDP Multicast Communication

Objective:
In this exercise, you will implement a simple one-to-many communication system using Python's `socket` library. You will complete two scripts: a **sender** that transmits a message to a multicast group, and a **receiver** that subscribes to that group, listens for the message, and sends back an acknowledgment.

Background:
In networking, there are three primary ways to send a message:

1. **Unicast:** One-to-one communication (e.g., browsing a website).
2. **Broadcast:** One-to-all communication on a local network segment (e.g., discovering devices). This can be inefficient as every host must process the packet.
3. **Multicast:** One-to-many communication. A sender sends a single packet to a special "multicast group" address. Only the computers that have explicitly subscribed (or "joined") that group will receive the packet. This is highly efficient for applications like live video streaming, online gaming, or distributing financial data, where the same data needs to go to multiple, specific clients simultaneously.

In this lab, you will build a multicast "broadcaster" and a "subscriber."

Scenario:
Imagine a news agency (the sender) needs to send an urgent bulletin to all its subscribed news terminals (the receivers) at once. Instead of sending an individual copy to each terminal, it will send one single message to the `224.1.1.1` multicast group. Any terminal that has joined this group will instantly receive the bulletin.

Your Task:
Your task is to complete the `multicast_sender_exercise.py` and `multicast_recv_exercise.py` files by filling in the code marked with `TODO` comments.

Part 1: The Receiver (`multicast_recv_exercise.py`)

The receiver is the more complex part, as it needs to tell the operating system that it's interested in receiving multicast traffic.

Key Steps:

1. **Create a UDP Socket:** The standard way to create a datagram socket.
2. **Bind the Socket:** The socket must be bound to a specific port to listen for incoming data. We bind to `''` (all network interfaces) so it can receive multicast packets regardless of which network card they arrive on.
3. **Join the Multicast Group:** This is the most important step. You must tell the operating system's IP layer to "add membership" for your socket to the multicast group. This requires a special `setsockopt` call with `IP_ADD_MEMBERSHIP`. The data for this option must be formatted correctly using `struct.pack()`.
4. **Receive and Respond:** Once subscribed, the receiver enters a loop to wait for data with `recvfrom()` and sends a unicast (one-to-one) acknowledgment back to the original sender.

Part 2: The Sender (`multicast_sender_exercise.py`)

The sender's job is simpler. It creates a message and sends it to the multicast group address, not to a specific client IP.

Key Steps:

1. **Create a UDP Socket:** Same as the receiver.
2. **Set the Time-To-Live (TTL):** Multicast packets have a TTL that determines how many network hops (routers) they can cross. A TTL of `1` is standard practice to ensure the packets do not leave the local network segment. This is configured using `setsockopt` with `IP_MULTICAST_TTL`.
3. **Send Data:** Use `sendto()` to send the message directly to the multicast group address and port.
4. **Wait for Acknowledgement:** The sender will then wait for a brief period to receive the unicast `ack` from any receivers that heard the message.

Instructions for Running:

You will need **two separate terminal windows** to run the receiver and the sender at the same

time.

1. **Terminal 1 (Receiver):**
   Start the receiver script first. It needs to be listening before the sender sends its message.

   ```
   python multicast_recv_exercise.py
   ```

   You should see the output: `waiting to receive message`.
2. **Terminal 2 (Sender):**
   In the second terminal, run the sender script.

   ```
   python multicast_sender_exercise.py
   ```

Expected Output:

- **In the Sender's Terminal:**

  ```
  sending "very important data"
  waiting to receive
  received "ack" from ('127.0.0.1', 10000)
  closing socket
  ```

- **In the Receiver's Terminal:**

  ```
  waiting to receive message
  received 19 bytes from ('127.0.0.1', <some_port>) : very important data
  sending acknowledgement to ('127.0.0.1', <some_port>)

  waiting to receive message
  ```