I socket in Python sono costruiti attorno a pochi comandi fondamentali, che vengono eseguiti in sequenza e sono leggermente diversi a seconda che si tratti del **Server** o del **Client**.
Ecco una spiegazione dei comandi principali che ho usato nei codici `server.py` e `client.py`, organizzati in una tabella chiara.

# 🔌 Comandi Fondamentali dei Socket in Python

| Comando Python | Server 🖥️ | Client 🐍 | Descrizione |
|---|---|---|---|
| `socket.socket(...)` | **Sì** | **Sì** | **Crea il Socket:** Inizializza l'oggetto socket. Abbiamo usato `socket.AF_INET` (per IPv4) e `socket.SOCK_STREAM` (per TCP, che garantisce la consegna dei dati). |
| `s.setsockopt(...)` | **Sì** | **No** | **Opzioni del Socket:** Imposta opzioni specifiche. Nel server, usiamo `socket.SO_REUSEADDR` per permettere il riavvio immediato (evitando l'errore "Address already in use"). |
| `s.bind((H, P))` | **Sì** | **No** | **Associazione:** Lega il socket all'**indirizzo IP (Host)** e alla **Porta** specificati. Il server "occupa" questa porta. |
| `s.listen()` | **Sì** | **No** | **Ascolto:** Mette il server in modalità di ascolto, pronto ad accettare le connessioni in arrivo. |
| `s.accept()` | **Sì** | **No** | **Accetta Connessione:** Si blocca e attende un client. Quando un client si connette, restituisce un **nuovo socket di connessione** (`conn`) e l'indirizzo del client (`addr`). |
| `s.connect((H, P))` | **No** | **Sì** | **Connessione:** Tenta di stabilire una connessione con il socket in ascolto all'indirizzo (Host) e alla Porta specificati. |
| `s.settimeout(T)` | **Sì/No** | **Sì** | **Timeout:** Imposta un tempo limite (T secondi) per le successive operazioni di blocco (come `connect`, `recv`, `send`). Nel Client è fondamentale per evitare blocchi infiniti. |
| `s.sendall(data)` | **Sì** | **Sì** | **Invio Dati:** Invia i dati (`data`) attraverso il socket connesso. `sendall` garantisce che tutti i byte vengano inviati. |
| `s.recv(bufsize)` | **Sì** | **Sì** | **Ricezione Dati:** Riceve dati dal socket. `bufsize` (1024 nel nostro caso) è il numero massimo di byte da ricevere in una singola operazione. Si blocca finché non arrivano dati o scade il timeout. |
| `s.close()` | **Sì** | **Sì** | **Chiusura:** Chiude la connessione del socket. Nei nostri esempi, usiamo `with conn:` e `with s:` che gestiscono la chiusura automaticamente. |

# 📝 Flusso Operativo (Semplificato)

## Server (Ascolta e Risponde)

1. **Crea** il socket (`socket.socket`).
2. **Imposta** l'opzione riutilizzo (`setsockopt`).
3. **Associa** l'indirizzo e la porta (`bind`).
4. **Ascolta** (`listen`).
5. **Accetta** la connessione (`accept`). <- **Qui il server aspetta.**
6. **Riceve** (`recv`).
7. **Invia** (`sendall`).
8. **Chiude** (gestito da `with`).

## Client (Connette e Scambia)

1. **Crea** il socket (`socket.socket`).
2. **Imposta** il timeout (`settimeout`).
3. **Connette** il socket al server (`connect`). <- **Qui il client aspetta.**
4. **Invia** (`sendall`).
5. **Riceve** (`recv`).
6. **Chiude** (gestito da `with`).