TCP Socket Programming Cheat Sheet

1. Import Statements

```
from socket import socket, AF_INET, SOCK_STREAM, SOL_SOCKET, SO_REUSEADDR, timeout, error
import struct  # For binary data packing/unpacking
import sys  # For command line arguments
```

2. Creating Sockets

Basic Socket Creation

```
# Create a TCP socket
client_socket = socket(AF_INET, SOCK_STREAM)
server_socket = socket(AF_INET, SOCK_STREAM)
```

Using Context Manager (Recommended)

```
# Automatically closes socket when done
with socket(AF_INET, SOCK_STREAM) as client:
    # Your socket code here
    pass
```

3. Client Socket Operations

Connect to Server

```
server_address = ('localhost', 10000)  # (host, port)
client_socket.connect(server_address)
```

Send Data

```
# Send string data (must encode to bytes)
message = "Hello Server"
client_socket.send(message.encode())

# Send all data (ensures complete transmission)
client_socket.sendall(message.encode())

# Send binary data (struct)
packed_data = struct.pack('I I 1s', num1, num2, op.encode())
client_socket.sendall(packed_data)
```

Receive Data

```
# Receive up to 1024 bytes
data = client_socket.recv(1024)

# Decode bytes to string
message = data.decode()

# Receive and decode in one line
message = client_socket.recv(1024).decode()

# Receive exact amount of data
data = client_socket.recv(struct_size)
```

Close Connection

```
client_socket.close()
```

4. Server Socket Operations

Bind to Address

```
server_address = ('', 10000)  # Empty string = all interfaces
# server_address = ('localhost', 10000)  # Only localhost
server_socket.bind(server_address)
```

Listen for Connections

```
server_socket.listen(1)  # Accept 1 pending connection
server_socket.listen(5)  # Accept up to 5 pending connections
server_socket.listen(0)  # Minimal backlog (demo purposes)
```

Accept Connections

```
client_socket, client_address = server_socket.accept()
print(f"Connected: {client_address}")
```

Server Socket Options

```
# Allow reusing address (prevents "Address already in use" errors)
server_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)

# Set timeout (useful for graceful shutdown)
server_socket.settimeout(1.0)
```

5. Complete Client Template

```
from socket import socket, AF_INET, SOCK_STREAM

server_address = ('localhost', 10000)

with socket(AF_INET, SOCK_STREAM) as client:
    client.connect(server_address)

    # Send data
    message = "Hello Server"
    client.sendall(message.encode())

    # Receive response
    response = client.recv(1024).decode()
    print(f"Received: {response}")
```

6. Complete Server Template

```
from socket import socket, AF_INET, SOCK_STREAM, SOL_SOCKET, SO_REUSEADDR

server_address = ('', 10000)

with socket(AF_INET, SOCK_STREAM) as server:
    server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    server.bind(server_address)
    server.listen(1)

    while True:
        try:
            client, client_addr = server.accept()
            print(f"Connected: {client_addr}")

            # Receive data
            data = client.recv(1024).decode()
            print(f"Received: {data}")

            # Send response
            response = "Hello Client"
            client.send(response.encode())

            client.close()

        except KeyboardInterrupt:
            break
```

7. Working with Binary Data (struct)

Packing Data (Client Side)

```
import struct

# Define format: I = unsigned int, 1s = 1-byte string
packer = struct.Struct('I I 1s')

# Pack data
num1, num2, operator = 5, 3, '+'
packed_data = packer.pack(num1, num2, operator.encode())

# Send packed data
client.sendall(packed_data)
```

Unpacking Data (Server Side)

```
import struct

# Define same format
unpacker = struct.Struct('I I 1s')

# Receive exact size
data = client.recv(unpacker.size)

# Unpack data
num1, num2, op_bytes = unpacker.unpack(data)
operator = op_bytes.decode()
```

8. Error Handling

Basic Error Handling

```
try:
    client.connect(server_address)
    # socket operations
except ConnectionRefusedError:
    print("Could not connect to server")
except timeout:
    print("Operation timed out")
except error:
    print("Socket error occurred")
```

Server with Timeout

```
server.settimeout(1.0)
while True:
    try:
        client, addr = server.accept()
        # handle client
    except timeout:
        pass  # Continue listening
    except KeyboardInterrupt:
        break
```

10. Common Patterns

Send-Receive Pattern

```
# Client
client.send("request".encode())
response = client.recv(1024).decode()

# Server
data = client.recv(1024).decode()
client.send("response".encode())
```

Message Protocol Pattern

```
# Define message types
MESSAGES = {
    'MOVE': 'Player should make a move',
    'WAIT': 'Wait for other player',
    'WIN': 'Player wins',
    'LOSS': 'Player loses'
}

# Send specific message
client.send("MOVE".encode())

# Handle different message types
message = client.recv(1024).decode()
if message == "MOVE":
    # Handle move request
elif message == "WAIT":
    # Handle wait state
```

11. Command Line Arguments

```
import sys

# Usage: python script.py <host> <port>
if len(sys.argv) != 3:
    print("Usage: python client.py <host> <port>")
    sys.exit(1)

host = sys.argv[1]
port = int(sys.argv[2])
```

12. Common Issues & Solutions

"Address already in use" Error

```
# Add this to server before bind()
server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
```

Partial Data Transmission

```
# Use sendall() instead of send() for complete transmission
client.sendall(data)

# For receiving exact amount:
def recv_exact(sock, size):
    data = b''
    while len(data) < size:
        packet = sock.recv(size - len(data))
        if not packet:
            break
        data += packet
    return data
```

String Encoding/Decoding

```
# Always encode strings before sending
message.encode()  # str -> bytes

# Always decode bytes after receiving
data.decode()  # bytes -> str
```

13. Debugging Tips

Print Debug Information

```
print(f"Connecting to {server_address}")
print(f"Sent: {message}")
print(f"Received: {response}")
print(f"Client connected from: {client_addr}")
```

Check Data Types

```
print(f"Data type: {type(data)}")
print(f"Data length: {len(data)}")
```

14. Key Reminders

- **Always encode strings** before sending: `message.encode()`
- **Always decode bytes** after receiving: `data.decode()`
- **Use** `sendall()` for reliable transmission
- **Close sockets** when done (or use `with` statement)
- **Handle exceptions** gracefully
- **Server address** `('', port)` accepts from any interface
- **Client address** `('localhost', port)` or `('127.0.0.1', port)` for local connection
- **Backlog parameter** in `listen()` controls pending connections queue size