1) How can you perform a port scan using Python?

Ans: - Performing a port scan using Python involves sending connection requests to a range of ports on a target system and analyzing the responses to determine which ports are open, closed, or filtered. Below are the steps to create a basic port scanner in Python using the **socket** library.

Input: -

```python
import socket

def port_scan(target, port_range):
    print(f"Scanning {target} for open ports in range {port_range[0]}-{port_range[1]}")

    for port in range(port_range[0], port_range[1] + 1):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)

        result = s.connect_ex((target, port))

        if result == 0:
            print(f"Port {port}: OPEN")
        s.close()

if __name__ == "__main__":
    target_host = "example.com"
    port_range = (20, 1024)
    port_scan(target_host, port_range)
```

Output: -

```
Scanning example.com for open ports in range 20-1024
Port 80: OPEN
Port 443: OPEN
```

2) Use the ipconfig /all command on a Windows machine to display detailed information about all network interfaces, including their MAC addresses.

Ans: -

```
PS C:\Users\subha\OneDrive\Documents\Desktop> ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : SUBHAYAN13
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Ethernet 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : ExpressVPN TAP Adapter
   Physical Address. . . . . . . . . : 00-FF-4A-BA-E0-23
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes

Unknown adapter Local Area Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : ExpressVPN TUN Driver
   Physical Address. . . . . . . . . :
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
```

```
Ethernet adapter Ethernet 3:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : VirtualBox Host-Only Ethernet Adapter
   Physical Address. . . . . . . . . : 0A-00-27-00-00-11
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::f170:3b25:d002:23ed%17(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.56.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 906625063
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2B-07-64-6D-7C-57-58-68-39-01
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter
   Physical Address. . . . . . . . . : 30-89-4A-38-E7-45
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #2
   Physical Address. . . . . . . . . : 32-89-4A-38-E7-44
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
```

```
Wireless LAN adapter Wi-Fi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) Wi-Fi 6E AX211 160MHz
   Physical Address. . . . . . . . . : 30-89-4A-38-E7-44
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes

Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Realtek Gaming GbE Family Controller
   Physical Address. . . . . . . . . : 7C-57-58-68-39-01
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
PS C:\Users\subha\OneDrive\Documents\Desktop>
```

3) Use the Arp -a command to display the ARP cache table on a Windows machine and identify the MAC addresses of devices on the local network.

Ans: -

```
PS C:\Users\subha\OneDrive\Documents\Desktop> arp -a

Interface: 192.168.194.102 --- 0x6
  Internet Address      Physical Address      Type
  192.168.194.151       8a-3e-ee-ba-e5-9c     dynamic
  192.168.194.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.56.1 --- 0x11
  Internet Address      Physical Address      Type
  192.168.56.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

4) Use the ping command to test connectivity between two devices on a local network by specifying their IP addresses. Interpret the output to identify any packet loss or latency issues.

Ans: -

Step-1: Open Command Prompt, type **ipconfig** (press Enter), and look for the "IPv4 Address" under my network adapter.

```
IPv4 Address. . . . . . . . . . . : 192.168.194.102
```

Step-2: In the Command Prompt type the following command, replacing <target_IP> with the actual IP address of the device you want to ping: ping <target_IP>

```
PS C:\Users\subha\OneDrive\Documents\Desktop> ping 192.168.194.102

Pinging 192.168.194.102 with 32 bytes of data:
Reply from 192.168.194.102: bytes=32 time=1ms TTL=128
Reply from 192.168.194.102: bytes=32 time<1ms TTL=128
Reply from 192.168.194.102: bytes=32 time<1ms TTL=128
Reply from 192.168.194.102: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.194.102:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

5) Use netstat -e command to show statistics about your network connection.

Ans: -

```
PS C:\Users\subha\OneDrive\Documents\Desktop> netstat -e
Interface Statistics

                           Received            Sent

Bytes                    1044848872        52115760
Unicast packets             833800          426352
Non-unicast packets             40            4424
Discards                         0               0
Errors                           0               0
Unknown protocols                0
```

6) Use the ping command with the -t option to continuously ping a remote host and observe the response times.

Ans: -

```
PS C:\Users\subha\OneDrive\Documents\Desktop> ping -t 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=216ms TTL=57
Reply from 8.8.8.8: bytes=32 time=59ms TTL=57
Reply from 8.8.8.8: bytes=32 time=69ms TTL=57
Reply from 8.8.8.8: bytes=32 time=551ms TTL=57
Reply from 8.8.8.8: bytes=32 time=62ms TTL=57
Reply from 8.8.8.8: bytes=32 time=66ms TTL=57
Reply from 8.8.8.8: bytes=32 time=60ms TTL=57
Reply from 8.8.8.8: bytes=32 time=223ms TTL=57
Reply from 8.8.8.8: bytes=32 time=232ms TTL=57
Reply from 8.8.8.8: bytes=32 time=62ms TTL=57
Reply from 8.8.8.8: bytes=32 time=65ms TTL=57
Reply from 8.8.8.8: bytes=32 time=63ms TTL=57
Reply from 8.8.8.8: bytes=32 time=97ms TTL=57
Reply from 8.8.8.8: bytes=32 time=99ms TTL=57
Reply from 8.8.8.8: bytes=32 time=62ms TTL=57
Reply from 8.8.8.8: bytes=32 time=63ms TTL=57
Reply from 8.8.8.8: bytes=32 time=62ms TTL=57
Reply from 8.8.8.8: bytes=32 time=56ms TTL=57

Ping statistics for 8.8.8.8:
    Packets: Sent = 18, Received = 18, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 56ms, Maximum = 551ms, Average = 120ms
Control-C
```

7) Use the netsh command to display the configuration of a network interface, including its MAC address, IP address, subnet mask, and default gateway.

Ans: -

```
PS C:\Users\subha\OneDrive\Documents\Desktop> netsh interface ip show config
```

```
Configuration for interface "Wi-Fi"
    DHCP enabled:                         Yes
    IP Address:                           192.168.194.102
    Subnet Prefix:                        192.168.194.0/24 (mask 255.255.255.0)
    Default Gateway:                      192.168.194.151
    Gateway Metric:                       0
    InterfaceMetric:                      55
    DNS servers configured through DHCP:  192.168.194.151
    Register with which suffix:           Primary only
    WINS servers configured through DHCP: None
```

```
PS C:\Users\subha\OneDrive\Documents\Desktop> getmac /v /fo list
```

```
Connection Name:  Wi-Fi
Network Adapter:  Intel(R) Wi-Fi 6E AX211 160MHz
Physical Address: 30-89-4A-38-E7-44
Transport Name:   \Device\Tcpip_{46745A83-CA63-43C7-96F2-EA706AB486A8}
```

```
PS C:\Users\subha\OneDrive\Documents\Desktop> ipconfig /all
```

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) Wi-Fi 6E AX211 160MHz
   Physical Address. . . . . . . . . : 30-89-4A-38-E7-44
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2409:4060:2dc5:8cab:6e24:1ec5:893a:6d3e(Preferred)
   Temporary IPv6 Address. . . . . . : 2409:4060:2dc5:8cab:820:9f74:563b:2966(Preferred)
   Link-local IPv6 Address . . . . . : fe80::d4a6:e63e:962e:a545%6(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.194.102(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : 18 June 2024 08:54:47
   Lease Expires . . . . . . . . . . : 18 June 2024 09:54:46
   Default Gateway . . . . . . . . . : fe80::883e:eeff:feba:e59c%6
                                       192.168.194.151
   DHCP Server . . . . . . . . . . . : 192.168.194.151
   DHCPv6 IAID . . . . . . . . . . . : 103844170
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2B-07-64-6D-7C-57-58-68-39-01
   DNS Servers . . . . . . . . . . . : 192.168.194.151
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

8) How can user create Multithreaded TCP Server that handles multiple clients?

Ans: -

Input: -

```python
import socket
import threading

def handle_client(client_socket, client_address):
    print(f"Accepted connection from {client_address}")

    while True:
        data = client_socket.recv(1024)
        if not data:
            break
        response = data
        client_socket.sendall(response)

    client_socket.close()
    print(f"Connection from {client_address} closed.")

def start_server(host, port):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(5)
    print(f"Server listening on {host}:{port}")

    while True:
        client_socket, client_address = server_socket.accept()
        client_thread = threading.Thread(
            target=handle_client, args=(client_socket, client_address)
        )
        client_thread.start()

if __name__ == "__main__":
    HOST = 'localhost'
    PORT = 12345

    start_server(HOST, PORT)
```

Output: -

```
Server listening on localhost:12345
Accepted connection from ('127.0.0.1', 62948)
Connection from ('127.0.0.1', 62948) closed.
Accepted connection from ('127.0.0.1', 62951)
Accepted connection from ('127.0.0.1', 62953)
Connection from ('127.0.0.1', 62951) closed.
Accepted connection from ('127.0.0.1', 62959)
Connection from ('127.0.0.1', 62953) closed.
Accepted connection from ('127.0.0.1', 62967)
Connection from ('127.0.0.1', 62959) closed.
Connection from ('127.0.0.1', 62967) closed.
Accepted connection from ('127.0.0.1', 62999)
Accepted connection from ('127.0.0.1', 63000)
Connection from ('127.0.0.1', 62999) closed.
```

9) Use the tracert command with the -d option to perform a traceroute without resolving hostnames to IP addresses, providing a faster output without DNS lookups.

Ans: -

```
PS C:\Users\subha\OneDrive\Documents\Desktop> tracert -d www.google.com

Tracing route to www.google.com [2404:6800:4002:819::2004]
over a maximum of 30 hops:

  1     2 ms     3 ms     1 ms  2409:4060:2dc5:8cab::ac
  2     *        *        *     Request timed out.
  3    67 ms    44 ms    38 ms  2405:200:351:eeee:20::964
  4    52 ms    39 ms    38 ms  2405:200:801:500::cc5
  5     *        *        *     Request timed out.
  6     *        *        *     Request timed out.
  7     *        *        *     Request timed out.
  8    78 ms    75 ms    82 ms  2001:4860:1:1::1a34
  9    93 ms    92 ms    83 ms  2001:4860:1:1::1a34
 10    98 ms    68 ms    78 ms  2404:6800:803d::1
 11    91 ms   379 ms    95 ms  2001:4860:0:1::1824
 12   212 ms    88 ms    90 ms  2001:4860:0:1::77dc
 13     *       90 ms     *     2001:4860:0:1::7599
 14    89 ms    78 ms    84 ms  2001:4860:0:1::54f7
 15   104 ms    58 ms    76 ms  2404:6800:4002:819::2004

Trace complete.
```

10) What command is used to trace the route to a remote host, displaying information about each hop along the path including packet loss and latency statistics?

Ans: - The command used to trace the route to a remote host, displaying information about each hop along the path including packet loss and latency statistics is **traceroute**.

11) How can you handle multiple clients simultaneously using sockets in Python?

Ans: - In Python, you can handle multiple clients using sockets by implementing a multi-threaded server or using asynchronous I/O.

For a multi-threaded server, you would create a new thread for each client connection, allowing each client to be handled concurrently.

Input: -

```
import socket
from _thread import *
import threading

print_lock = threading.Lock()

def threaded(c):
    while True:
        data = c.recv(1024)
        if not data:
            print('Bye')
            print_lock.release()
            break
        data = data[::-1]
        c.send(data)
    c.close()

def Main():
    host = ""
    port = 12345
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    print("socket binded to port", port)
    s.listen(5)
    print("socket is listening")

    while True:
        c, addr = s.accept()
        print_lock.acquire()
        print('Connected to :', addr[0], ':', addr[1])
        start_new_thread(threaded, (c,))
    s.close()

if __name__ == "__main__":
    Main()
```

Output: -

```
socket binded to port 12345
socket is listening
Connected to : 127.0.0.1 : 55783
Bye
Connected to : 127.0.0.1 : 55786
Bye
Connected to : 127.0.0.1 : 55807
Bye
Connected to : 127.0.0.1 : 55808
Bye
Connected to : 127.0.0.1 : 55830
Bye
Connected to : 127.0.0.1 : 55831
Bye
Connected to : 127.0.0.1 : 55834
Bye
```

12) Develop a C /Python code snippet to create a TCP server that listens for incoming connections on port 8080 and prints "Connection accepted" when a client connects.

Ans: - Python implementation →

Input: -

```
import socket

def Main():
    host = ""
    port = 8080
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    print("Socket binded to port", port)
    s.listen(5)
    print("Socket is listening")

    while True:
        c, addr = s.accept()
        print("Connection accepted")
        c.close()

if __name__ == "__main__":
    Main()
```

Output: -

```
Socket binded to port 8080
Socket is listening
Connection accepted
Connection accepted
Connection accepted
Connection accepted
Connection accepted
Connection accepted
Connection accepted
Connection accepted
Connection accepted
```

13) Write a C /Python program that creates a UDP client-server pair that sends and receives data using UDP sockets.

Ans: - Python implementation →

Code: UDP Server

```
import socket

def udp_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind(('localhost', 12345))

    print("UDP server up and listening")

    while True:
        message, client_address = server_socket.recvfrom(1024)
        print(f"Received message from {client_address}: {message.decode()}")

        response_message = "Message received"
        server_socket.sendto(response_message.encode(), client_address)

if __name__ == "__main__":
    udp_server()
```

Output:

```
UDP server up and listening
Received message from ('127.0.0.1', 58293): Hello, UDP server!
```

Code: UPD Client

```python
import socket

def udp_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_address = ('localhost', 12345)
    message = "Hello, UDP server!"

    try:
        client_socket.sendto(message.encode(), server_address)

        response, _ = client_socket.recvfrom(1024)
        print(f"Received response from server: {response.decode()}")

    finally:
        client_socket.close()

if __name__ == "__main__":
    udp_client()
```

Output:

```
Received response from server: Message received
```

14) Write a C /Python program to implement the parity generator code from a given pattern.

Ans: - Python implementation →

Input:

```python
def generate_parity(data):
    # Count the number of 1's in the data
    count_ones = data.count('1')

    # If count of 1's is odd, return '1' for even parity
    if count_ones % 2 != 0:
        return '1'
    else:
        return '0'

binary_data = input("Enter binary data: ")
parity_bit = generate_parity(binary_data)
print(f"The parity bit for {binary_data} is {parity_bit}")
```

Output:

```
Enter binary data:  011100
The parity bit for 011100 is 1
```

15) Write a C /Python program to implement the parity checker code from a given pattern.

Ans: - Python implementation →

Input:

```python
def check_parity(data):
    # Count the number of 1's in the data
    count_ones = data.count('1')

    # If count of 1's is even, the parity bit is correct
    if count_ones % 2 == 0:
        return True
    else:
        return False

binary_data = input("Enter binary data with parity bit: ")
is_valid = check_parity(binary_data)

if is_valid:
    print(f"The binary data {binary_data} has correct parity.")
else:
    print(f"The binary data {binary_data} has incorrect parity.")
```

Output:

i)
```
Enter binary data with parity bit:  11010
The binary data 11010 has incorrect parity.
```

ii)
```
Enter binary data with parity bit:  11011
The binary data 11011 has correct parity.
```

16) Write a C /Python program for counter generator.

Ans: -

Python implementation →

Input:

```
def counter(start, stop, step=1):
    current = start
    while current < stop:
        yield current
        current += step

start = int(input("Enter the starting number: "))
stop = int(input("Enter the ending number: "))
step = int(input("Enter the step size (default is 1): "))

# Using the counter generator
print(f"Generated sequence from {start} to {stop-1} with step {step}:")
for num in counter(start, stop, step):
    print(num)
```

Output:

```
Enter the starting number:  2
Enter the ending number:  20
Enter the step size (default is 1):  2
Generated sequence from 2 to 19 with step 2:
2
4
6
8
10
12
14
16
18
```

17) Write a Python program to implement CRC.

Ans: - Python implementation →

Input:

```
def crc_remainder(input_bitstring, polynomial_bitstring, initial_filler):

    polynomial_bitstring = polynomial_bitstring.lstrip('0')
    initial_padding = initial_filler * (len(polynomial_bitstring) - 1)
    input_padded = list(input_bitstring + initial_padding)
    while '1' in input_padded[:len(input_bitstring)]:
        cur_shift = input_padded.index('1')
        for i in range(len(polynomial_bitstring)):
            input_padded[cur_shift + i] = str(int(polynomial_bitstring[i] != input_padded[cur_shift + i]))
    return ''.join(input_padded)[len(input_bitstring):]

def crc_check(input_bitstring, polynomial_bitstring, crc_check_bitstring):

    polynomial_bitstring = polynomial_bitstring.lstrip('0')
    initial_padding = crc_check_bitstring
    input_padded = list(input_bitstring + initial_padding)
    while '1' in input_padded[:len(input_bitstring)]:
        cur_shift = input_padded.index('1')
        for i in range(len(polynomial_bitstring)):
            input_padded[cur_shift + i] = str(int(polynomial_bitstring[i] != input_padded[cur_shift + i]))
    return '1' not in ''.join(input_padded)[len(input_bitstring):]
```

```
if __name__ == "__main__":
    input_bitstring = input("Enter binary data (message): ")
    polynomial_bitstring = input("Enter CRC polynomial in binary form (e.g., 1101): ")
    initial_filler = input("Enter initial CRC value (e.g., 000): ")

    crc_remainder_bits = crc_remainder(input_bitstring, polynomial_bitstring, initial_filler)
    print(f"CRC remainder (checksum): {crc_remainder_bits}")

    received_crc = input("Enter received CRC (checksum): ")

    crc_correct = crc_check(input_bitstring, polynomial_bitstring, received_crc)
    if crc_correct:
        print("CRC check passed. Data is correct.")
    else:
        print("CRC check failed. Data may be corrupted.")
```

Output:

i)

```
Enter binary data (message):  101010
Enter CRC polynomial in binary form (e.g., 1101):  1101
Enter initial CRC value (e.g., 000):  000
CRC remainder (checksum): 011000000
Enter received CRC (checksum):  011
CRC check passed. Data is correct.
```

ii)

```
Enter binary data (message):  101010
Enter CRC polynomial in binary form (e.g., 1101):  1101
Enter initial CRC value (e.g., 000):  000
CRC remainder (checksum): 011000000
Enter received CRC (checksum):  101
CRC check failed. Data may be corrupted.
```

18) Write a C /Python program using stream oriented server using TCP port no. 3456.

Ans: - Python implementation →

Input:

```python
import socket

def start_server(host='localhost', port=3456):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server is listening on {host}:{port}")

    while True:
        print("Waiting for a connection...")
        connection, client_address = server_socket.accept()

        try:
            print(f"Connection from {client_address}")
            while True:
                # Receive the length of the message first
                data_length = connection.recv(4)
                if not data_length:
                    break
                data_length = int.from_bytes(data_length, 'big')

                # Receive the actual message data
                data = connection.recv(data_length).decode()
                print(f"Received: {data}")

                if data:
                    print("Sending data back to the client")
                    connection.sendall(data_length.to_bytes(4, 'big') + data.encode())
                else:
                    print("No data from client, closing connection")
                    break
        finally:
            connection.close()

if __name__ == "__main__":
    start_server()
```

Output:

```
Server is listening on localhost:3456
Waiting for a connection...
Connection from ('127.0.0.1', 60173)
Received: This is a test message from the client
Sending data back to the client
Waiting for a connection...
```

19) Write a C /Python program using stream oriented client using TCP port no. 3456.

Ans: - Python implementation →

Input:

```python
import socket

def start_client(server_host='localhost', server_port=3456):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))

    try:
        message = 'This is a test message from the client'
        print(f"Sending: {message}")

        message_length = len(message)
        client_socket.sendall(message_length.to_bytes(4, 'big') + message.encode())

        data_length = client_socket.recv(4)
        data_length = int.from_bytes(data_length, 'big')

        data = client_socket.recv(data_length).decode()
        print(f"Received: {data}")
    finally:
        client_socket.close()

if __name__ == "__main__":
    start_client()
```

Output:

```
Sending: This is a test message from the client
Received: This is a test message from the client
```

---------- End ----------