SRM INSTITUTE OF SCIENCE & TECHNOLOGY

Kattankulathur

Chennai

Name	➢	**GAURAV GUPTA**
Subject	➢	Advanced Programming practice
Section	➢	W2
Roll No.	➢	RA2211026010284
Title	➢	Assignment Week 11

Assignment

Week 11

Q1) Develop a simple Python program of TCP, client that can connect to the server and client can send a "Hello, Server!" message to the server.

CODE:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    # Connect to the server
    client_socket.connect(server_address)

    # Send a message to the server
    message = "Hello, Server!"
    client_socket.sendall(message.encode())

    # Receive a response from the server
    response = client_socket.recv(1024)
    print(f"Server says: {response.decode()}")
```

```
except ConnectionRefusedError:
    print("Connection to the server failed. Make sure the server is running.")

finally:
    # Close the socket
    client_socket.close()
```

Q2. Develop a Python program that allows the TCP client to send a list of numbers to the server. The server should calculate and return the sum of the numbers to the client.

CODE

Server:

```
import socket
import pickle

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)
print("Server is listening for connections...")
```

```
while True:
    # Accept a connection
    client_socket, client_address = server_socket.accept()
    print(f"Connection established with {client_address}")

    try:
        # Receive the list of numbers from the client
        data = client_socket.recv(1024)
        numbers = pickle.loads(data)
        result = sum(numbers)

        # Send the result back to the client
        response = str(result)
        client_socket.sendall(response.encode())
    except Exception as e:
        print(f"Error: {e}")
    finally:
        # Close the client socket
        client_socket.close()
```

Client:

```
import socket
import pickle

# Define the server address and port
server_address = ('localhost', 12345)
```

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



try:
    # Connect to the server
    client_socket.connect(server_address)

    # Prepare a list of numbers
    numbers = [1, 2, 3, 4, 5]

    # Serialize and send the list to the server
    data = pickle.dumps(numbers)
    client_socket.sendall(data)

    # Receive the result from the server
    response = client_socket.recv(1024)
    result = response.decode()
    print(f"Server responded with the sum: {result}")
except ConnectionRefusedError:
    print("Connection to the server failed. Make sure the server is
running.")
except Exception as e:
    print(f"Error: {e}")
finally:
    # Close the socket
```

client_socket.close()

Q3) Create a Python UDP client that sends a "UDP Message" packet to a UDP server. Demonstrate the sending and receiving of the packet.

CODE:

UDP SERVER:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(server_address)

print("UDP server is running...")

while True:
    data, client_address = server_socket.recvfrom(1024)
    print(f"Received data from {client_address}: {data.decode()}")
    response = "Message received and acknowledged."
    server_socket.sendto(response.encode(), client_address)
```

UDP CLIENT:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

try:
    # Send a UDP message to the server
    message = "UDP Message"
    client_socket.sendto(message.encode(), server_address)

    # Receive a response from the server
    response, server_address = client_socket.recvfrom(1024)
    print(f"Received response from {server_address}: {response.decode()}")
except Exception as e:
    print(f"Error: {e}")
finally:
    # Close the socket
    client_socket.close()
```

Q4) Create a Python UDP client that sends a random number to the UDP server. The server should check if the number is even or odd and send the result back to the client.

CODE:

UDP SERVER:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(server_address)

print("UDP server is running...")

while True:
    data, client_address = server_socket.recvfrom(1024)
    received_number = int(data.decode())

    if received_number % 2 == 0:
        response = "Even"
    else:
        response = "Odd"


server_socket.sendto(response.encode(), client_address)
```

UDP CLIENT:

```
import socket
import random

# Define the server address and port
server_address = ('localhost', 12345)
```

```
# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

try:
    # Generate a random number
    random_number = random.randint(1, 100)
    print(f"Sending random number to the server: {random_number}")
    client_socket.sendto(str(random_number).encode(), server_address)

    # Receive the result from the server
    response, server_address = client_socket.recvfrom(1024)
    print(f"Received result from the server: {response.decode()}")
except Exception as e:
    print(f"Error: {e}")
finally:
    # Close the socket
    client_socket.close()
```


Q5) Write a Python program to create a UDP server that listens on port 54321. Ensure the server can receive UDP packets from clients.

CODE

Ensure the server can receive UDPpackets from clients.

```
import socket

# Define the server address and port
server_address = ('localhost', 54321)

# Create a socket and bind it to the server address
```

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(server_address)

print("UDP server is running...")

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

Q6) Write a python program to Generate Fibonacci Sequence.

CODE

UDP SERVER:

```
import socket

# Define the server address and port
server_address = ('localhost', 54321)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(server_address)

print("UDP server is running...")

while True:
    data, client_address = server_socket.recvfrom(1024)
    received_message = data.decode()

    if received_message == "UDP Message":
```

```
        response = "Message received and acknowledged."
        server_socket.sendto(response.encode(), client_address)
        print(f"Acknowledgment sent to {client_address}")
    else:
        print(f"Received data from {client_address}: {received_message}")
```

UDP CLIENT:

```
import socket

# Define the server address and port
server_address = ('localhost', 54321)

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

try:
    # Send a "UDP Message" packet to the server
    message = "UDP Message"
    client_socket.sendto(message.encode(), server_address)

    # Receive the acknowledgment from the server
    response, server_address = client_socket.recvfrom(1024)

    print(f"Received acknowledgment from {server_address}:
{response.decode()}")
except Exception as e:
    print(f"Error: {e}")
finally:
    # Close the socket
```

```
client_socket.close()
```

Q7) Implement a Python program that calculates and displays the time taken for a TCP client to connect to the server and receive a response. Measure the time elapsed in seconds.

CODE:

```
import socket
import time

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)
print("TCP server is running...")

while True:
    # Accept a connection
    client_socket, client_address = server_socket.accept()
    print(f"Connection established with {client_address}")

    # Simulate some work or processing
    time.sleep(2)

    # Send a response to the client
    response = "Hello, Client!"
    client_socket.sendall(response.encode())

    # Close the client socket
    client_socket.close()
```

CLIENT:

```
import socket
import time

# Define the server address and port
server_address = ('localhost', 12345)


# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    # Record the start time
    start_time = time.time()

    # Connect to the server
    client_socket.connect(server_address)

    # Receive a response from the server
    response = client_socket.recv(1024)

    # Record the end time
    end_time = time.time()

    print(f"Received response: {response.decode()}")

    # Calculate and display the time taken in seconds
    elapsed_time = end_time - start_time
    print(f"Time taken: {elapsed_time:.3f} seconds")
except Exception as e:
    print(f"Error: {e}")
finally:
    # Close the socket
    client_socket.close()
```


Q8) Create a TCP server that echoes back any message it receives from a client. Develop a Python client to send messages to the server and display the echoed response.

Code:

TCP SERVER:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)
print("TCP server is running...")

while True:

 # Accept a connection
   client_socket, client_address = server_socket.accept()
   print(f"Connection established with {client_address}")

   try:
      while True:
         # Receive data from the client
         data = client_socket.recv(1024)
         if not data:
            break

         # Echo the data back to the client
         client_socket.sendall(data)
```

```
    except Exception as e:
        print(f"Error: {e}")
    finally:
        # Close the client socket
        client_socket.close()
```

TCP CLIENT:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    # Connect to the server
    client_socket.connect(server_address)

    while True:
        message = input("Enter a message (or 'exit' to quit): ")
        if message == 'exit':
            break

        # Send the message to the server
        client_socket.sendall(message.encode())

        # Receive and display the echoed response
        response = client_socket.recv(1024)
        print(f"Server echoed: {response.decode()}")
```

```
except Exception as e:
    print(f"Error: {e}")
finally:
    # Close the socket
    client_socket.close()
```

Q 9) Develop a simple Python program that sends a small text file from a TCP client to a TCP server. Confirm that the file is received and saved correctly.

CODE:

TCP SERVER:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)
print("TCP server is running...")

while True:
    # Accept a connection
    client_socket, client_address = server_socket.accept()
    print(f"Connection established with {client_address}")
```

```
    try:
        # Receive the file content from the client
        file_content = b''
        while True:
            data = client_socket.recv(1024)
            if not data:
                break
            file_content += data

        # Save the received file
        with open("received_file.txt", "wb") as file:
            file.write(file_content)

        print("File received and saved as 'received_file.txt'")
    except Exception as e:
        print(f"Error: {e}")
    finally:
        # Close the client socket
        client_socket.close()
```

TCP CLIENT:

```
import socket

# Define the server address and port
server_address = ('localhost', 12345)

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
```

```
    # Connect to the server
    client_socket.connect(server_address)

    # Read the text file
    with open("sample_file.txt", "rb") as file:
        file_content = file.read()

    # Send the file content to the server
    client_socket.sendall(file_content)


    print("File sent to the server")
except Exception as e:
    print(f"Error: {e}")
finally:
    # Close the socket
    client_socket.close()
```


Q 10) Write a Python program to receive UDP packets and display their content. Simulatesending UDPpackets from a separate client program.

CODE:

UDP SERVER:

```
import socket

# Define the server address and port
server_address = ('localhost', 54321)

# Create a socket and bind it to the server address
```

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(server_address)

print("UDP server is running...")

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

UDP CLIENT:

```
import socket

# Define the server address and port
server_address = ('localhost', 54321)

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

try:
    while True:
        message = input("Enter a message to send (or 'exit' to quit): ")
        if message == 'exit':
            break

        # Send the message to the server
        client_socket.sendto(message.encode(), server_address)
except Exception as e:
    print(f"Error: {e}")
```

```
finally:
    # Close the socket
    client_socket.close()
```