1. Write a short discussion explaining all four models.

A. Remote Procedure Call (RPC)

Concept: RPC allows a program to call a procedure (function) on another computer as if it were local.

How it works:The client sends a request to a server to execute a specific function. The server executes it and returns the result.

Example use:Email systems, database updates, client-server applications.

Advantage:Simple for programmers — hides the details of network communication.

Disadvantage:Both client and server must be active at the same time; tightly coupled.

B. Remote Object Invocation (RMI)

Concept:
RMI is similar to RPC but uses objects instead of procedures.
It allows one Java object to call methods on another Java object located on a remote machine.

How it works: RMI uses *stub* (client proxy) and *skeleton* (server-side handler) to send method calls and return results.

Example use:Distributed Java applications, cloud-based services, or chat servers.

Advantage:Supports object-oriented design; works well with Java.

Disadvantage: Java-specific; not easily compatible with other languages.

C. Message-Oriented Middleware (MOM)

Concept:MOM enables communication between distributed systems using messages stored in queues.

It’s asynchronous — the sender and receiver don’t need to be running at the same time.

Example: Java Message Service (JMS), IBM MQ.

Advantage: Reliable, asynchronous, and loosely coupled communication.

Disadvantage:Slower than direct communication (due to message queuing).

D. Stream-Oriented Communication

Concept:This model is used when data must be transmitted continuously and in real time (e.g., video/audio streaming).

How it works:
Data is sent as a stream (sequence of packets) that must arrive on time.
Example:Zoom meetings, YouTube, live video streaming.

Advantage:Supports real-time media transfer.

Disadvantage:Needs high bandwidth and Quality of Service (QoS) guarantees.

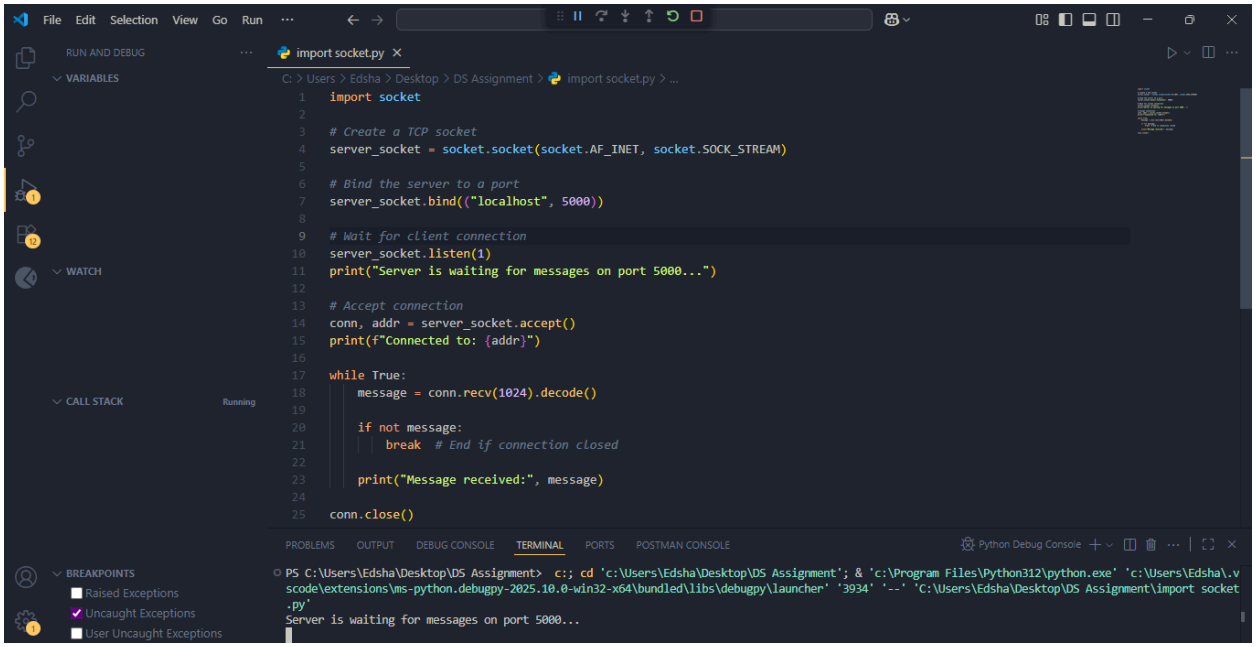
2,Choose one model and implement it in Java or Python.

1. Server Code (Receiver)

```
import socket

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

# Bind the server to a port
server_socket.bind(("localhost", 5000))

# Wait for client connection
server_socket.listen(1)
print("Server is waiting for messages on port 5000...")

# Accept connection
conn, addr = server_socket.accept()
```

```
print(f"Connected to: {addr}")

while True:
    message = conn.recv(1024).decode()

    if not message:
        break  # End if connection closed

    print("Message received:", message)

conn.close()
```

- **Result**

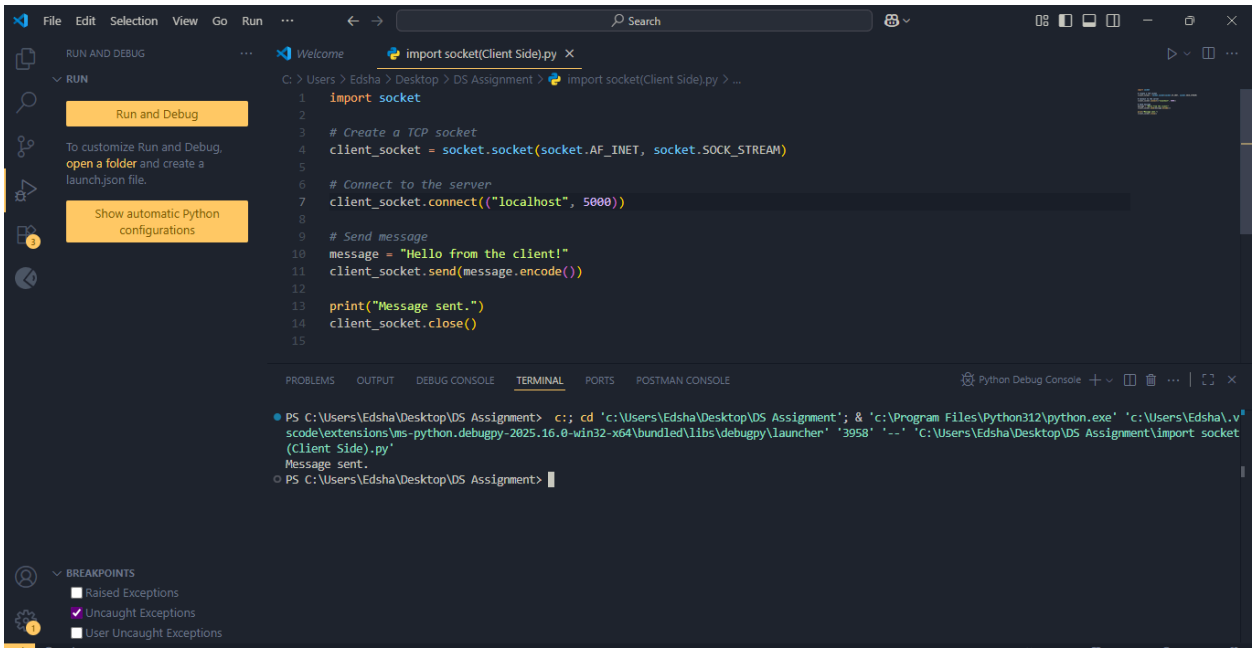


2. Client Code (Sender)

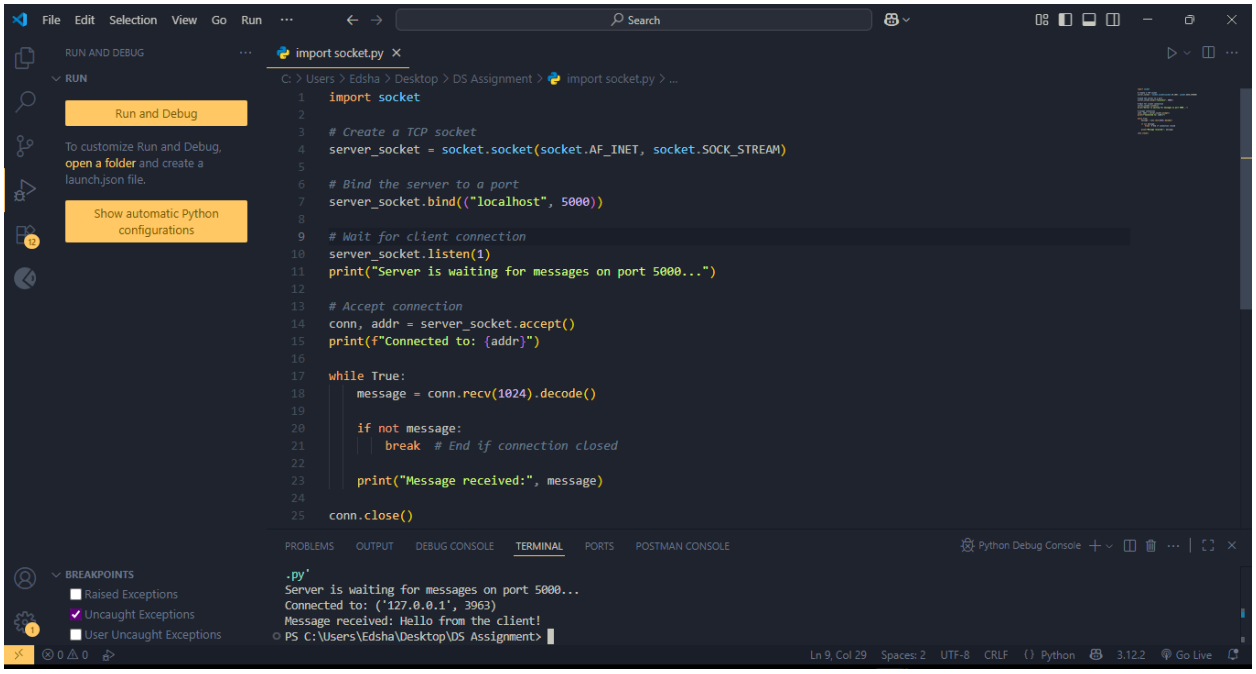
```
import socket

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

# Connect to the server
client_socket.connect(("localhost", 5000))

# Send message
message = "Hello from the client!"
client_socket.send(message.encode())

print("Message sent.")
client_socket.close()
```

- **Result**

Client Side



Server Side


File Edit Selection View Go Run
Search
RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show automatic Python configurations
import socket.py
C: > Users > Edsha > Desktop > DS Assignment > import socket.py > ...
import socket
# Create a TCP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Bind the server to a port
server_socket.bind(("localhost", 5000))
# Wait for client connection
server_socket.listen(1)
print("Server is waiting for messages on port 5000...")
# Accept connection
conn, addr = server_socket.accept()
print(f"Connected to: {addr}")
while True:
message = conn.recv(1024).decode()
if not message:
break # End if connection closed
print("Message received:", message)
conn.close()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
Python Debug Console
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
User Uncaught Exceptions
.py'
Server is waiting for messages on port 5000...
Connected to: ('127.0.0.1', 3963)
Message received: Hello from the client!
PS C:\Users\Edsha\Desktop\DS Assignment>
Ln 9, Col 29 Spaces: 2 UTF-8 CRLF Python 3.12.2 Go Live