

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()

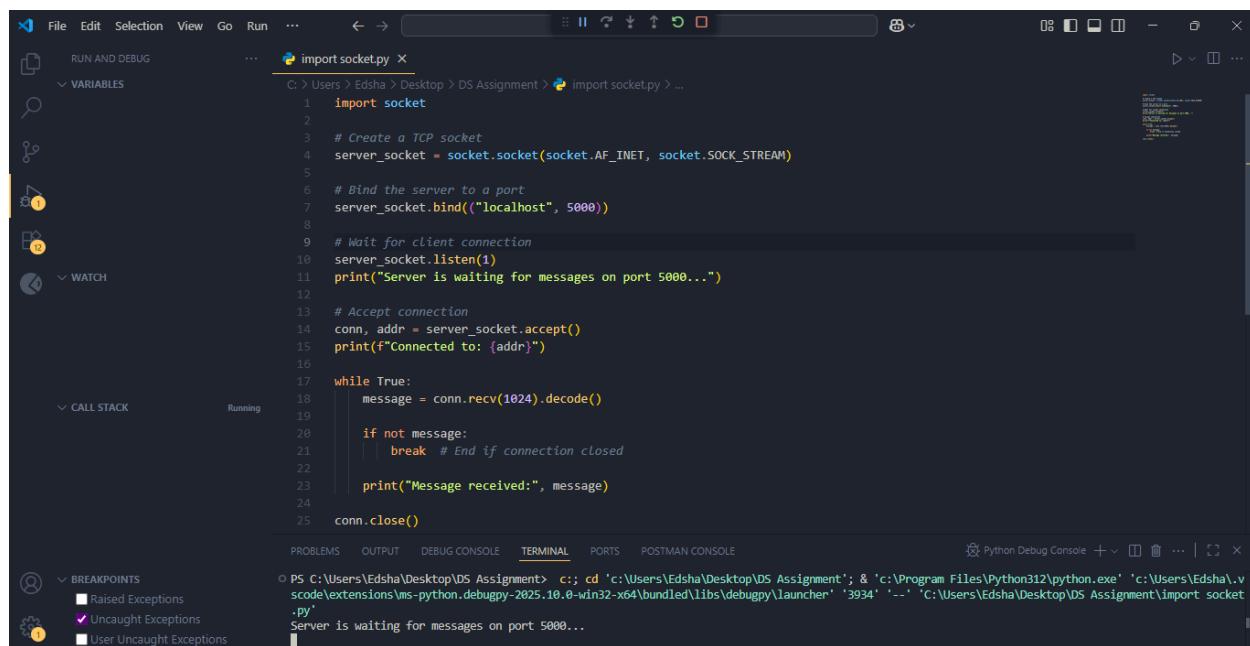
    if not message:
        break # End if connection closed

    print("Message received:", message)

conn.close()

```

● Result



The screenshot shows a Python debugger interface with the following details:

- File Path:** C:\Users\Edsha\Desktop>DS Assignment> import socket.py > ...
- Code View:** The code is displayed in the main pane, showing the server socket creation, binding to port 5000, listening for connections, accepting them, and printing the connected address.
- Run Status:** The code is running, indicated by the "Running" status in the call stack panel.
- Breakpoints:** A breakpoint is set at line 15, which is the first line of the while loop.
- Call Stack:** The call stack shows the current execution path starting from the main function.
- Variables:** A variables panel is visible on the left, showing the current values of variables like `server_socket` and `addr`.
- Output:** The terminal tab shows the command used to run the script and the message "Server is waiting for messages on port 5000...".

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

The screenshot shows a Python code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** Search
- Run and Debug Panel:** RUN AND DEBUG, RUN, Run and Debug (highlighted), Show automatic Python configurations.
- Code Editor:** Welcome, import socket(Client Side).py
C:\Users\Edsha\Desktop\DS Assignment> import socket(Client Side).py > ...

```
1 import socket
2
3 # Create a TCP socket
4 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5
6 # Connect to the server
7 client_socket.connect(("localhost", 5000))
8
9 # Send message
10 message = "Hello from the client!"
11 client_socket.send(message.encode())
12
13 print("Message sent.")
14 client_socket.close()
15
```
- Terminal Tab:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (highlighted), PORTS, POSTMAN CONSOLE
Python Debug Console
- Terminal Output:**
 - PS C:\Users\Edsha\Desktop\DS Assignment> c:; cd 'c:\Users\Edsha\Desktop\DS Assignment'; & 'c:\Program Files\Python312\python.exe' 'c:\Users\Edsha\vscode\extensions\ms-python.debugpy-2025.16.0-win32-x64\bundled\libs\debugpy\launcher' '3958' '--' 'C:\Users\Edsha\Desktop\DS Assignment\import_socket (Client Side).py'
 - Message sent.
 - PS C:\Users\Edsha\Desktop\DS Assignment> |
- Breakpoints:** BREAKPOINTS, Raised Exceptions, Uncaught Exceptions, User Uncaught Exceptions

Server Side

The screenshot shows the Microsoft Visual Studio Code interface with a Python file named `socket.py` open. The code implements a simple TCP server that listens on port 5000, waits for a message from a client, and then prints it to the terminal.

```
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...")

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

    if not message:
        break # End if connection closed

    print("Message received:", message)

conn.close()
```

The terminal tab shows the server's output:

```
.py'
Server is waiting for messages on port 5000...
Connected to: ('127.0.0.1', 3963)
Message received: Hello from the client!
```

The status bar at the bottom indicates the file is saved at `C:\Users\Edsha\Desktop\DS Assignment>`.