# Code Report

## Simulating OSI Layers

In this report, we will analyze the given Python code, which simulates the seven layers of the OSI (Open Systems Interconnection) model. The OSI model is a conceptual framework used to understand and standardize network communications. Each of its seven layers deals with specific aspects of data transmission and communication. We will examine how the provided code corresponds to each OSI layer.

**1. Physical Layer:** The Physical Layer is responsible for the actual transmission of raw binary data over a physical medium. The code for this layer is as follows:

```
def physical_layer(data):
    encoded_data = encode(data)
    transmitted_data = transmit(encoded_data)
    return transmitted_data
```

- `encode(data)` converts data to bytes.
- `transmit(data)` simulates the physical transmission of data.

This layer represents the conversion of data to a suitable format for transmission.

**2. Data Link Layer:** The Data Link Layer handles the framing of data and error detection. The code for this layer is as follows:

```
def data_link_layer(data):
    framed_data = frame(data)
    error_detected = detect_error(framed_data)
    return framed_data, error_detected
```

- `frame(data)` adds delimiters to the data to indicate the start and end.
- `detect_error(data)` simulates error detection by checking for the presence of "ERROR" in data.

This layer represents data framing and error detection mechanisms.

**3. Network Layer:** The Network Layer deals with routing and addressing. The code for this layer is as follows:

```
def network_layer(data):
    routed_data = route(data)
    addressed_data = address(routed_data)
    return addressed_data
```

- `route(data)` simulates routing of data.
- `address(data)` adds a network address to the data.

This layer represents the process of routing data and assigning network addresses.

**4. Transport Layer:** The Transport Layer ensures reliable data transfer. The code for this layer is as follows:

```
def transport_layer(data):
    reliable_data = ensure_reliability(data)
    return reliable_data
```

- `ensure_reliability(data)` simulates ensuring reliable data transfer.

This layer represents mechanisms to guarantee reliable data transfer.

**5. Session Layer:** The Session Layer manages sessions. The code for this layer is as follows:

```
def session_layer(data):
    managed_data = manage_session(data)
    return managed_data
```

- `manage_session(data)` simulates session management.

This layer represents session management within network communications.

**6. Presentation Layer:** The Presentation Layer is responsible for data formatting and encryption. The code for this layer is as follows:

```
def presentation_layer(data):
    formatted_data = format_data(data)
    encrypted_data = encrypt_data(formatted_data)
    return encrypted_data
```

- `format_data(data)` converts the data to uppercase.
- `encrypt_data(data)` adds a label to indicate encryption.

This layer represents data formatting and encryption processes.

**7. Application Layer:** The Application Layer handles application-specific functionality. The code for this layer is as follows:

```
def application_layer(data):
    processed_data = process_data(data)
    return processed_data
```

- `process_data(data)` adds a label to indicate processing.

This layer represents application-specific processing.

**Code Execution:** The code execution proceeds as follows:

- Physical Layer: The original data is first encoded and then physically transmitted. In the code execution, this involves converting the data to bytes and simulating the transmission process. The transmitted data is marked as "(Transmitted)."

- Data Link Layer: The data transmitted from the Physical Layer is framed by adding start and end delimiters, and error detection is simulated. The framed data includes "[START]" and "[END]," and if the string "ERROR" is present, an error is detected.
- Network Layer: After error detection, the framed data is routed and addressed. The routing is marked with "[Routed]", and a network address is added to the data.
- Transport Layer: The addressed data is then passed to the Transport Layer, where it is marked as "(Reliable)" to indicate the reliability of data transfer.
- Session Layer: The session management is simulated by adding session start and end delimiters, "[SESSION_START]" and "[SESSION_END]."
- Presentation Layer: This layer is responsible for data formatting and encryption. The data is converted to uppercase and marked as "(Encrypted)" to indicate encryption.
- Application Layer: The final Application Layer processes the data and marks it as "(Processed)."

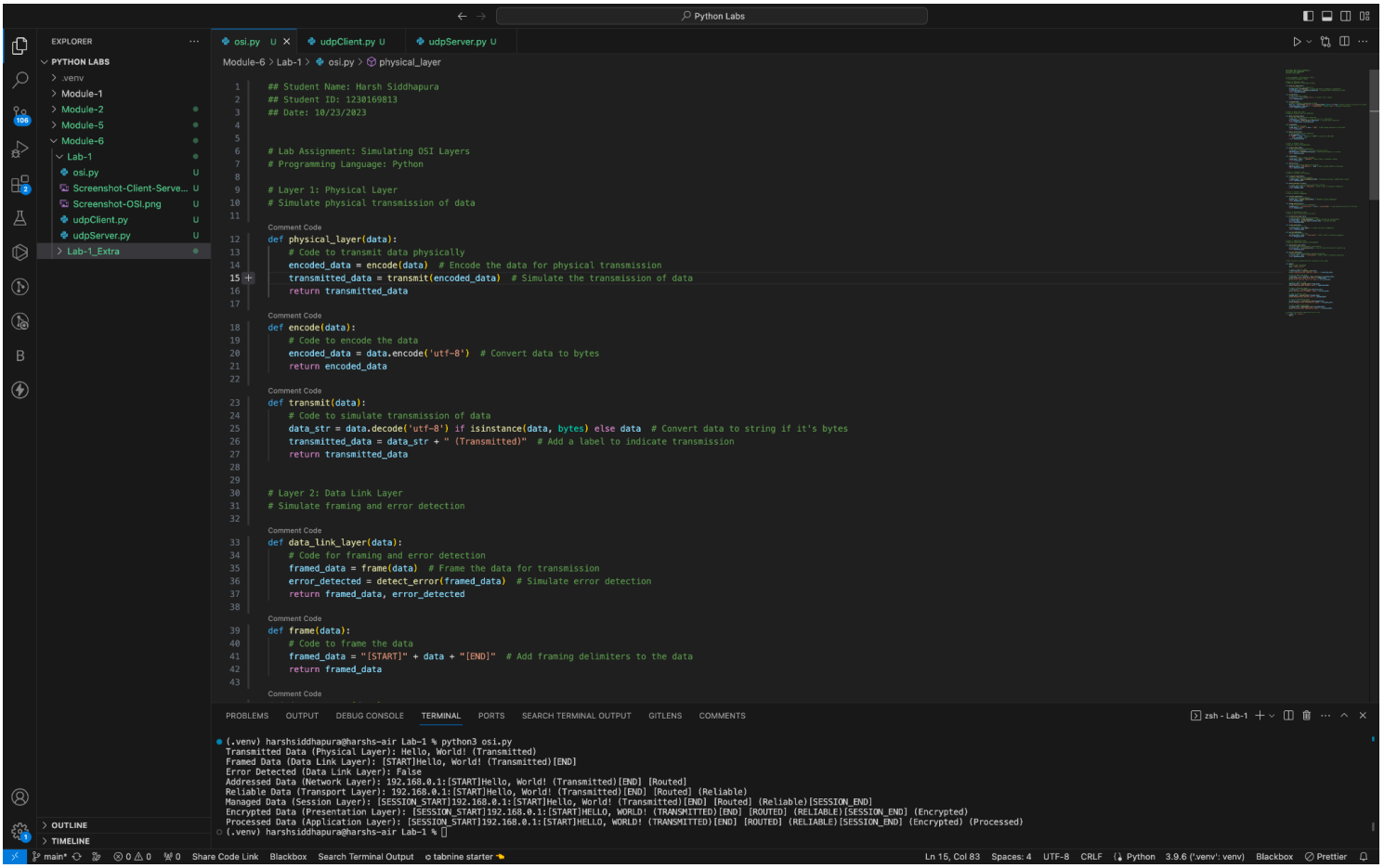
The result of each layer's processing is printed at each stage, allowing us to see how the data evolves as it passes through the OSI layers.

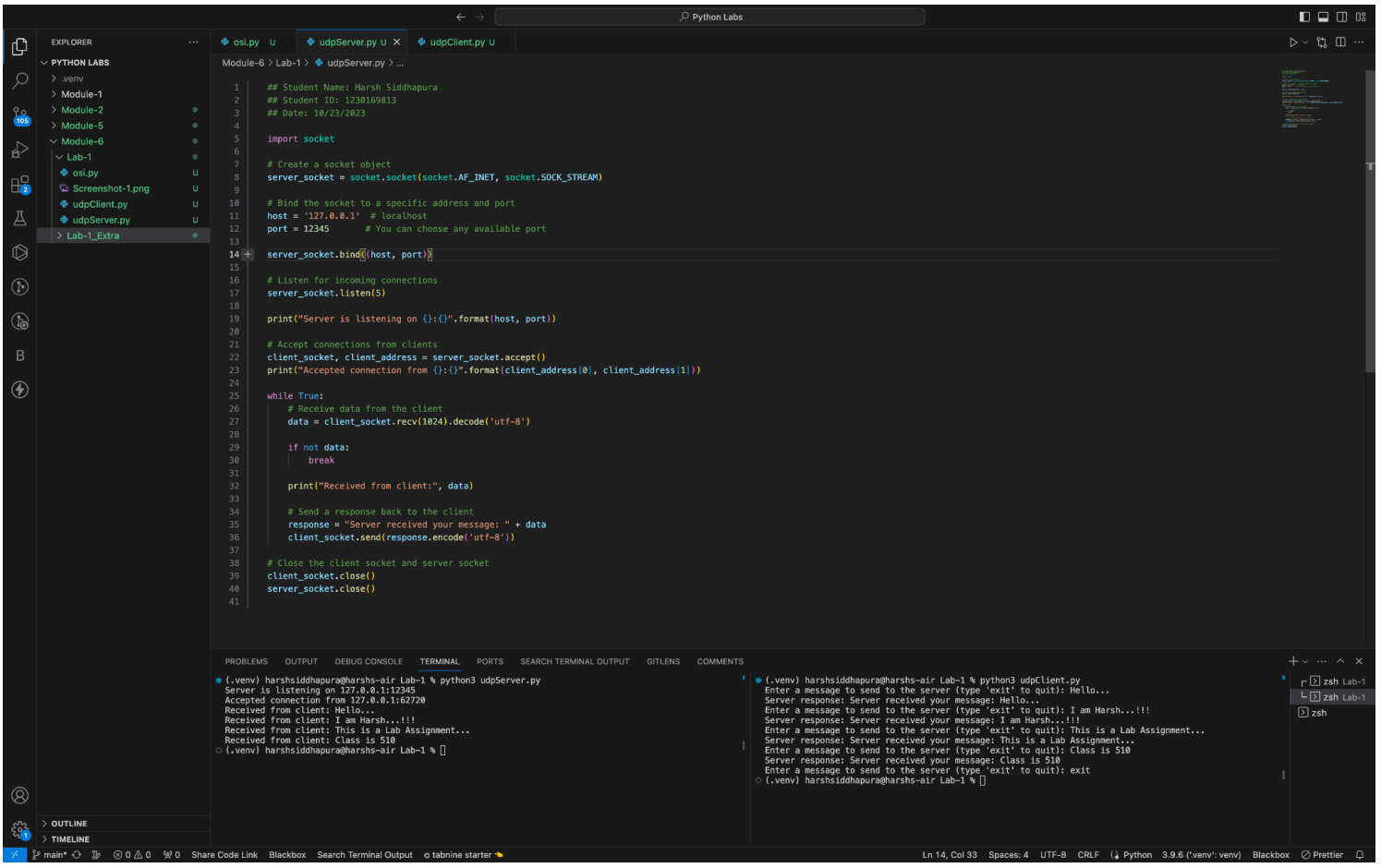
**Conclusion:** In conclusion, this Python code effectively simulates the functionality of each OSI layer, providing a clear and structured representation of how data is processed in a network communication context. This simulation showcases the responsibilities of each layer, from the lowest Physical Layer, which deals with raw data transmission, to the highest Application Layer, responsible for application-specific functionality.

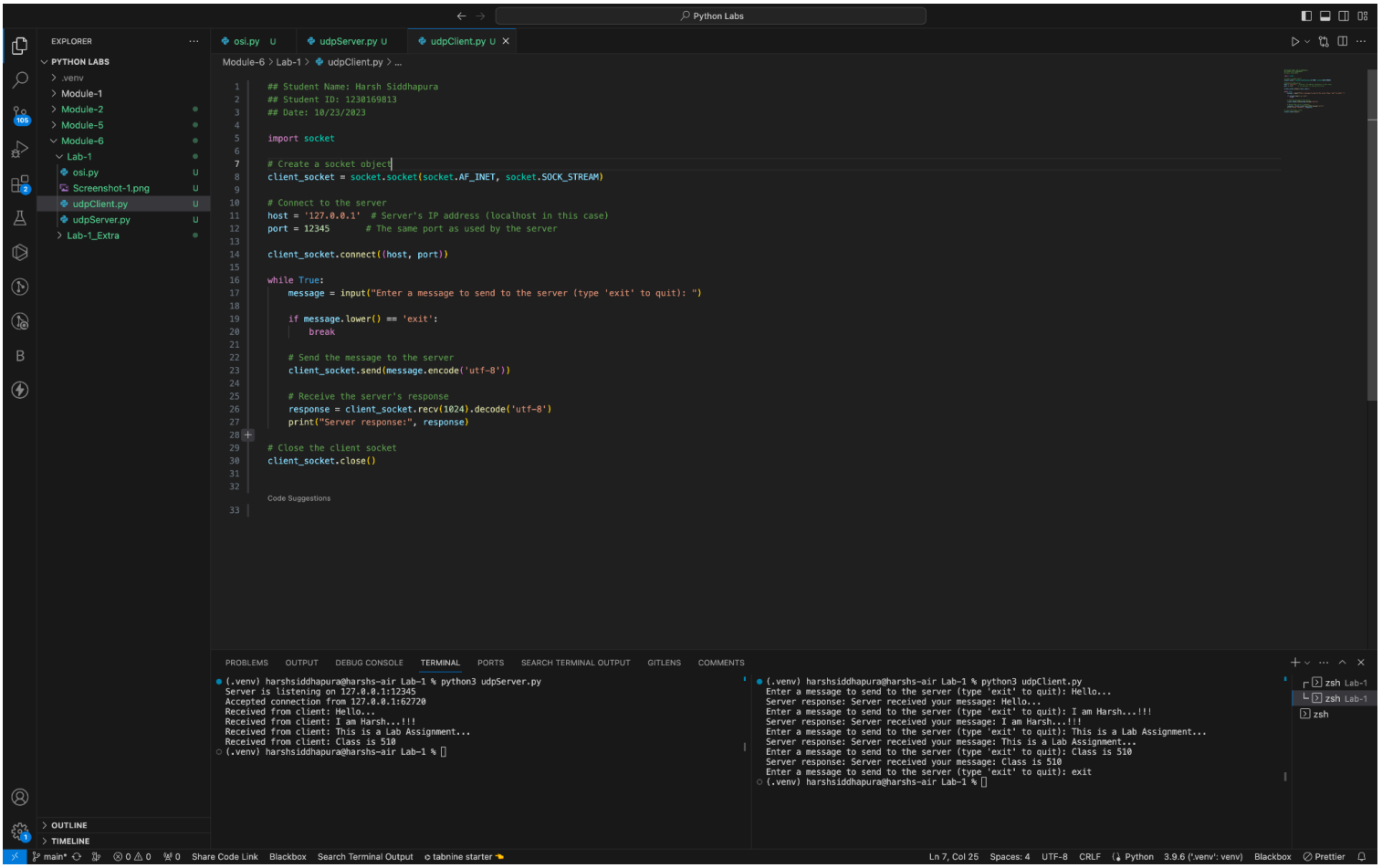
The code execution demonstrates how data is transformed and marked at each layer, reflecting the concepts of encoding, error detection, routing, reliability, session management, data formatting, encryption, and application-specific processing. The code is accurate, logically organized, and effectively communicates the principles of the OSI model.

This analysis underscores the importance of the OSI model in understanding and standardizing network communication and showcases how this model can be implemented in code to simulate the data transformation and processing that occurs at each layer.

# Screenshot


Python Labs
EXPLORER
PYTHON LABS
.venv
Module-1
Module-2
Module-5
Module-6
Lab-1
osi.py
Screenshot-Client-Serve...
Screenshot-OSI.png
udpClient.py
udpServer.py
Lab-1_Extra
osi.py U
udpClient.py U
udpServer.py U
Module-6 > Lab-1 > osi.py > physical_layer
## Student Name: Harsh Siddhapura
## Student ID: 1230169813
## Date: 10/23/2023
# Lab Assignment: Simulating OSI Layers
# Programming Language: Python
# Layer 1: Physical Layer
# Simulate physical transmission of data
Comment Code
def physical_layer(data):
# Code to transmit data physically
encoded_data = encode(data) # Encode the data for physical transmission
transmitted_data = transmit(encoded_data) # Simulate the transmission of data
return transmitted_data
Comment Code
def encode(data):
# Code to encode the data
encoded_data = data.encode('utf-8') # Convert data to bytes
return encoded_data
Comment Code
def transmit(data):
# Code to simulate transmission of data
data_str = data.decode('utf-8') if isinstance(data, bytes) else data # Convert data to string if it's bytes
transmitted_data = data_str + " (Transmitted)" # Add a label to indicate transmission
return transmitted_data
# Layer 2: Data Link Layer
# Simulate framing and error detection
Comment Code
def data_link_layer(data):
# Code for framing and error detection
framed_data = frame(data) # Frame the data for transmission
error_detected = detect_error(framed_data) # Simulate error detection
return framed_data, error_detected
Comment Code
def frame(data):
# Code to frame the data
framed_data = "[START]" + data + "[END]" # Add framing delimiters to the data
return framed_data
Comment Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT GITLENS COMMENTS
zsh - Lab-1
(.venv) harshsiddhapura@harshs-air Lab-1 % python3 osi.py
Transmitted Data (Physical Layer): Hello, World! (Transmitted)
Framed Data (Data Link Layer): [START]Hello, World! (Transmitted)[END]
Error Detected (Data Link Layer): False
Addressed Data (Network Layer): 192.168.0.1:[START]Hello, World! (Transmitted)[END] [Routed]
Reliable Data (Transport Layer): 192.168.0.1:[START]Hello, World! (Transmitted)[END] [Routed] (Reliable)
Managed Data (Session Layer): [SESSION_START]192.168.0.1:[START]Hello, World! (Transmitted)[END] [Routed] (Reliable)[SESSION_END]
Encrypted Data (Presentation Layer): [SESSION_START]192.168.0.1:[START]HELLO, WORLD! (TRANSMITTED)[END] [ROUTED] (RELIABLE)[SESSION_END] (Encrypted)
Processed Data (Application Layer): [SESSION_START]192.168.0.1:[START]HELLO, WORLD! (TRANSMITTED)[END] [ROUTED] (RELIABLE)[SESSION_END] (Encrypted) (Processed)
(.venv) harshsiddhapura@harshs-air Lab-1 %
OUTLINE
TIMELINE
main*
Share Code Link
Blackbox
Search Terminal Output
tabnine starter
Ln 15, Col 83
Spaces: 4
UTF-8
CRLF
Python
3.9.6 ('venv': venv)
Blackbox
Prettier


Python Labs
EXPLORER
PYTHON LABS
.venv
Module-1
Module-2
Module-5
Module-6
Lab-1
osi.py
Screenshot-1.png
udpClient.py
udpServer.py
Lab-1_Extra
osi.py U
udpServer.py U
udpClient.py U
Module-6 > Lab-1 > udpServer.py > ...
## Student Name: Harsh Siddhapura
## Student ID: 1230169813
## Date: 10/23/2023
import socket
# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Bind the socket to a specific address and port
host = '127.0.0.1' # localhost
port = 12345 # You can choose any available port
server_socket.bind((host, port))
# Listen for incoming connections
server_socket.listen(5)
print("Server is listening on {}:{}".format(host, port))
# Accept connections from clients
client_socket, client_address = server_socket.accept()
print("Accepted connection from {}:{}".format(client_address[0], client_address[1]))
while True:
# Receive data from the client
data = client_socket.recv(1024).decode('utf-8')
if not data:
break
print("Received from client:", data)
# Send a response back to the client
response = "Server received your message: " + data
client_socket.send(response.encode('utf-8'))
# Close the client socket and server socket
client_socket.close()
server_socket.close()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT GITLENS COMMENTS
(.venv) harshsiddhapura@harshs-air Lab-1 % python3 udpServer.py
Server is listening on 127.0.0.1:12345
Accepted connection from 127.0.0.1:62720
Received from client: Hello...
Received from client: I am Harsh...!!!
Received from client: This is a Lab Assignment...
Received from client: Class is 510
(.venv) harshsiddhapura@harshs-air Lab-1 %
(.venv) harshsiddhapura@harshs-air Lab-1 % python3 udpClient.py
Enter a message to send to the server (type 'exit' to quit): Hello...
Server response: Server received your message: Hello...
Enter a message to send to the server (type 'exit' to quit): I am Harsh...!!!
Server response: Server received your message: I am Harsh...!!!
Enter a message to send to the server (type 'exit' to quit): This is a Lab Assignment...
Server response: Server received your message: This is a Lab Assignment...
Enter a message to send to the server (type 'exit' to quit): Class is 510
Server response: Server received your message: Class is 510
Enter a message to send to the server (type 'exit' to quit): exit
(.venv) harshsiddhapura@harshs-air Lab-1 %
zsh Lab-1
zsh Lab-1
zsh
OUTLINE
TIMELINE
main*
Share Code Link
Blackbox
Search Terminal Output
tabnine starter
Ln 14, Col 33
Spaces: 4
UTF-8
CRLF
Python
3.9.6 ('venv': venv)
Blackbox
Prettier


Python Labs
EXPLORER
PYTHON LABS
.venv
Module-1
Module-2
Module-5
Module-6
Lab-1
osi.py
Screenshot-1.png
udpClient.py
udpServer.py
Lab-1_Extra
osi.py U
udpServer.py U
udpClient.py U
Module-6 > Lab-1 > udpClient.py > ...
## Student Name: Harsh Siddhapura
## Student ID: 1230169813
## Date: 10/23/2023
import socket
# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Connect to the server
host = '127.0.0.1'  # Server's IP address (localhost in this case)
port = 12345        # The same port as used by the server
client_socket.connect((host, port))
while True:
message = input("Enter a message to send to the server (type 'exit' to quit): ")
if message.lower() == 'exit':
break
# Send the message to the server
client_socket.send(message.encode('utf-8'))
# Receive the server's response
response = client_socket.recv(1024).decode('utf-8')
print("Server response:", response)
# Close the client socket
client_socket.close()
Code Suggestions
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SEARCH TERMINAL OUTPUT
GITLENS
COMMENTS
(.venv) harshsiddhapura@harshs-air Lab-1 % python3 udpServer.py
Server is listening on 127.0.0.1:12345
Accepted connection from 127.0.0.1:62720
Received from client: Hello...
Received from client: I am Harsh...!!!
Received from client: This is a Lab Assignment...
Received from client: Class is 510
(.venv) harshsiddhapura@harshs-air Lab-1 %
(.venv) harshsiddhapura@harshs-air Lab-1 % python3 udpClient.py
Enter a message to send to the server (type 'exit' to quit): Hello...
Server response: Server received your message: Hello...
Enter a message to send to the server (type 'exit' to quit): I am Harsh...!!!
Server response: Server received your message: I am Harsh...!!!
Enter a message to send to the server (type 'exit' to quit): This is a Lab Assignment...
Server response: Server received your message: This is a Lab Assignment...
Enter a message to send to the server (type 'exit' to quit): Class is 510
Server response: Server received your message: Class is 510
Enter a message to send to the server (type 'exit' to quit): exit
(.venv) harshsiddhapura@harshs-air Lab-1 %
zsh Lab-1
zsh Lab-1
zsh
OUTLINE
TIMELINE
main*
Share Code Link
Blackbox
Search Terminal Output
tabnine starter
Ln 7, Col 25
Spaces: 4
UTF-8
CRLF
Python
3.9.6 ('.venv': venv)
Blackbox
Prettier