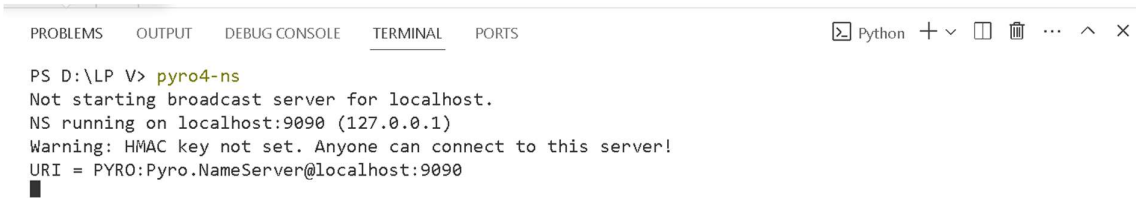
1. Implement multi-threaded client/server Process communication using RMI.

pyro4-ns ---> tarminal command to run sever

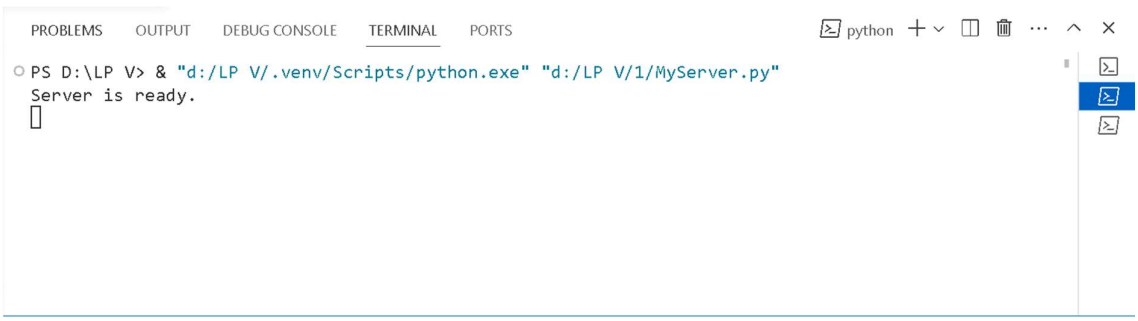
Running server



MyServer.py

```
import Pyro4

@Pyro4.expose
class MyRemoteClass(object):
    def addition(self, x, y):
        return x + y

    def mult(self, x, y):
        return x * y

def main():
    daemon = Pyro4.Daemon()
    ns = Pyro4.locateNS()
    uri = daemon.register(MyRemoteClass)
    ns.register("MyRemoteClass", uri)
    print("Server is ready.")
    daemon.requestLoop()

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

Output



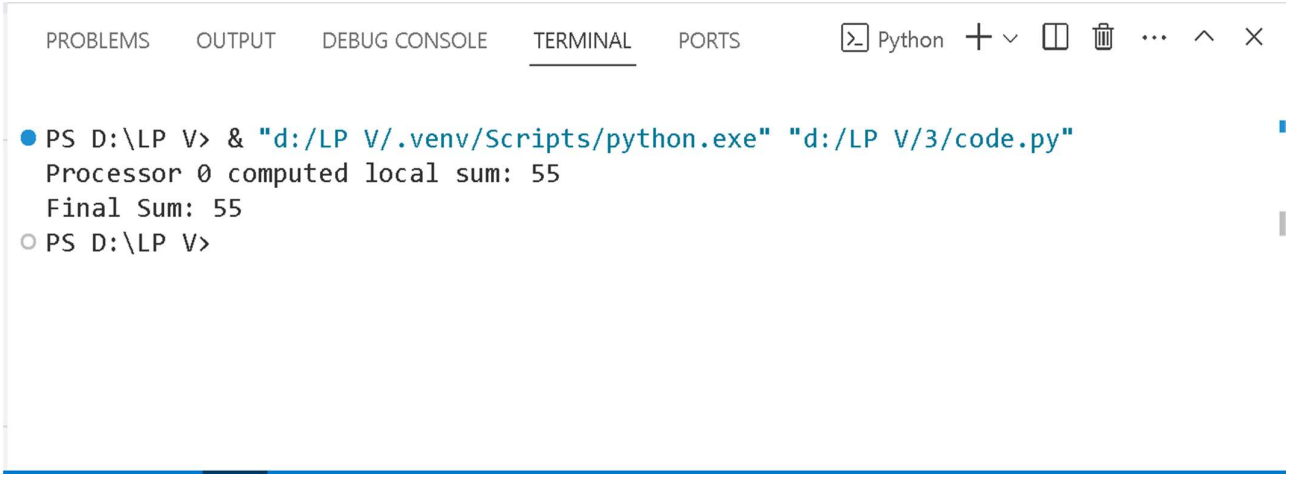
MyClient.py

```python
import Pyro4

def main():
    try:
        uri = "PYRONAME:MyRemoteClass"
        obj = Pyro4.Proxy(uri)
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))
        print("The Addition is:", obj.addition(a, b))
        print("The Multiplication is:", obj.mult(a, b))
    except Exception as e:
        print("Error:", e)

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

3. Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors.

```
from mpi4py import MPI

def distribute_array(arr, comm):
    rank = comm.Get_rank()
    size = comm.Get_size()
    chunk_size = len(arr) // size
    start = rank * chunk_size
    end = start + chunk_size if rank < size - 1 else len(arr)
    return arr[start:end]

def compute_sum(arr):
    return sum(arr)

def main():
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    size = comm.Get_size()

    # Define the array to be summed (Assuming same array on all processors)
    array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Distribute the array among processors
    local_array = distribute_array(array, comm)

    # Compute local sum
    local_sum = compute_sum(local_array)

    # Gather all local sums on root process (rank 0)
    all_sums = comm.gather(local_sum, root=0)

    # Display intermediate sums calculated at different processors
    print("Processor", rank, "computed local sum:", local_sum)

    # Root process combines sums to get the final result
    if rank == 0:
        final_sum = sum(all_sums)
        print("Final Sum:", final_sum)

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

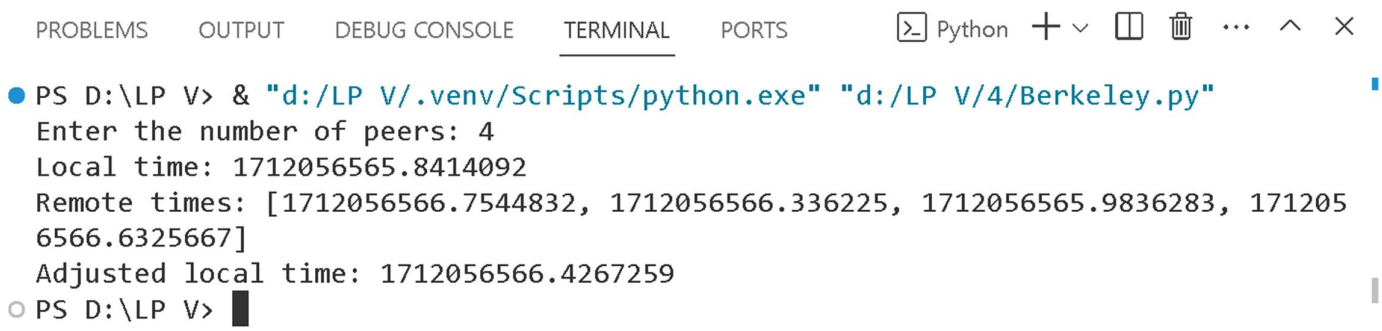
Output


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
PS D:\LP V> & "d:/LP V/.venv/Scripts/python.exe" "d:/LP V/3/code.py"
Processor 0 computed local sum: 55
Final Sum: 55
PS D:\LP V>

4. Implement Berkeley algorithm for clock synchronization.

```
import time
import random

# Function to calculate the clock offset
def calculate_offset(remotes):
    local_time = time.time()
    offsets = [remote - local_time for remote in remotes]
    return sum(offsets) / len(offsets)

# Function to synchronize clocks using the Berkeley algorithm
def synchronize_clocks():
    num_peers = int(input("Enter the number of peers: "))
    local_time = time.time()

    # Simulate remote clocks with random offsets
    remote_times = [local_time + random.uniform(-1, 1) for _ in
range(num_peers)]
    print("Local time:", local_time)
    print("Remote times:", remote_times)

    # Calculate the clock offset
    offset = calculate_offset(remote_times)

    # Adjust local clock
    adjusted_time = local_time + offset
    print("Adjusted local time:", adjusted_time)

# Execute the clock synchronization
synchronize_clocks()
```

Output



5. Implement token ring based mutual exclusion algorithm.

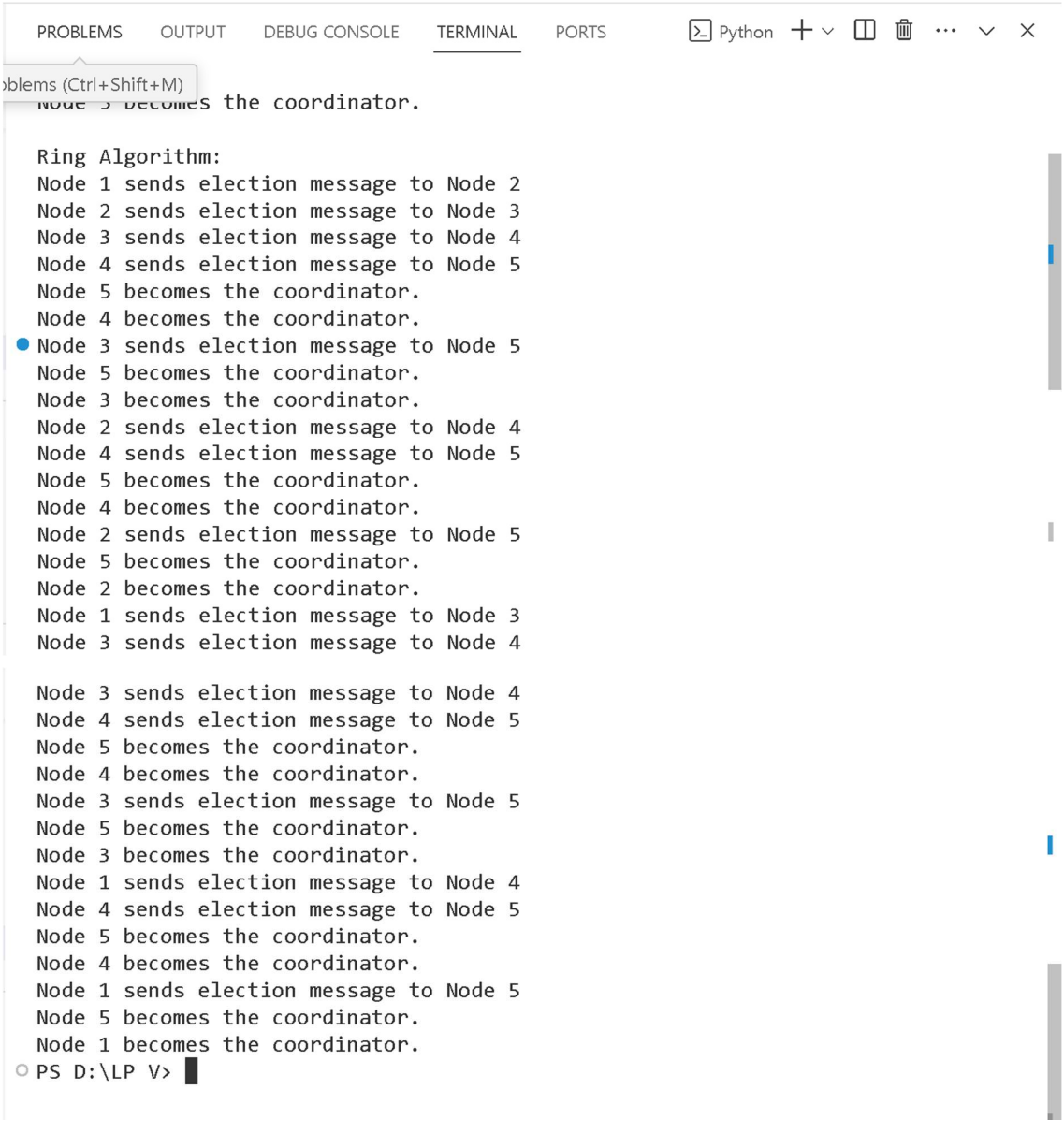
```python
class TokenRing:
    def __init__(self, num_nodes):
        self.num_nodes = num_nodes
        self.token = 0

    def send_data(self, sender, receiver, data):
        print("Token passing:", end="")
        for i in range(self.token, sender):
            print(f" {i % self.num_nodes}->", end="")
        print(f" {sender}")
        print(f"Sender {sender} sending data: {data}")

        for i in range(sender + 1, receiver):
            print(f"Data {data} forwarded by {i}")

        print(f"Receiver {receiver} received data: {data}\n")
        self.token = sender

if __name__ == "__main__":
    num_nodes = int(input("Enter the number of nodes: "))
    token_ring = TokenRing(num_nodes)

    while True:
        sender = int(input("Enter sender: "))
        receiver = int(input("Enter receiver: "))
        data = input("Enter data: ")

        token_ring.send_data(sender, receiver, data)

        send_again = input("Do you want to send again? (yes/no): ")
        if send_again.lower() != "yes":
            break
```

Output

6. Implement Bully and Ring algorithm for leader election.

```
class Node:
    def __init__(self, node_id):
        self.id = node_id
        self.is_coordinator = False

    def initiate_election(self, nodes):
        for node in nodes:
            if node.id > self.id:
                print(f"Node {self.id} sends election message to Node
{node.id}")
                node.start_election(nodes)
        self.is_coordinator = True
        print(f"Node {self.id} becomes the coordinator.")

    def start_election(self, nodes):
        for node in nodes:
            if node.id > self.id:
                print(f"Node {self.id} sends election message to Node
{node.id}")
                node.start_election(nodes)
        self.is_coordinator = True
        print(f"Node {self.id} becomes the coordinator.")

if __name__ == "__main__":
    # Create nodes
    nodes = [Node(i) for i in range(1, 6)]

    # Simulate Bully Algorithm
    print("Bully Algorithm:")
    # Node with highest ID starts the election
    nodes[-1].initiate_election(nodes)

    # Simulate Ring Algorithm
    print("\nRing Algorithm:")
    # Node with lowest ID starts the election
    nodes[0].start_election(nodes)
```

Output

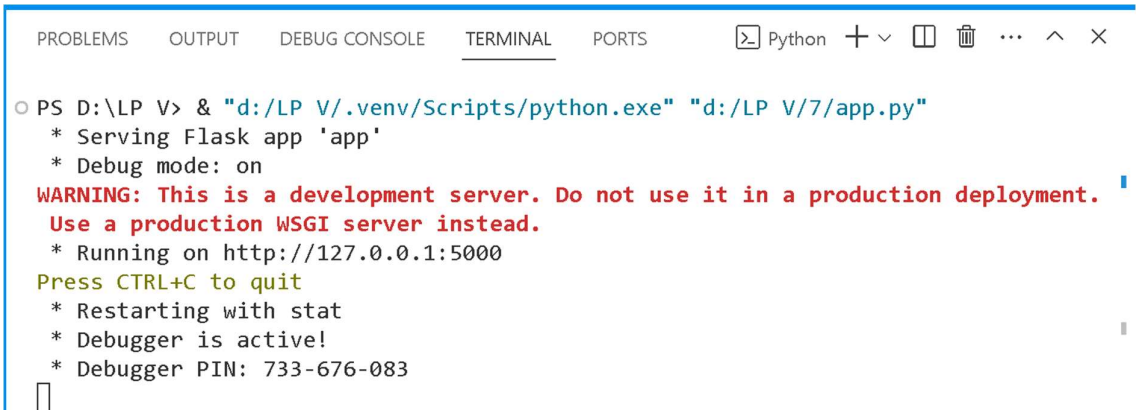

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
blems (Ctrl+Shift+M)
Node 3 becomes the coordinator.
Ring Algorithm:
Node 1 sends election message to Node 2
Node 2 sends election message to Node 3
Node 3 sends election message to Node 4
Node 4 sends election message to Node 5
Node 5 becomes the coordinator.
Node 4 becomes the coordinator.
Node 3 sends election message to Node 5
Node 5 becomes the coordinator.
Node 3 becomes the coordinator.
Node 2 sends election message to Node 4
Node 4 sends election message to Node 5
Node 5 becomes the coordinator.
Node 4 becomes the coordinator.
Node 2 sends election message to Node 5
Node 5 becomes the coordinator.
Node 2 becomes the coordinator.
Node 1 sends election message to Node 3
Node 3 sends election message to Node 4
Node 3 sends election message to Node 4
Node 4 sends election message to Node 5
Node 5 becomes the coordinator.
Node 4 becomes the coordinator.
Node 3 sends election message to Node 5
Node 5 becomes the coordinator.
Node 3 becomes the coordinator.
Node 1 sends election message to Node 4
Node 4 sends election message to Node 5
Node 5 becomes the coordinator.
Node 4 becomes the coordinator.
Node 1 sends election message to Node 5
Node 5 becomes the coordinator.
Node 1 becomes the coordinator.
PS D:\LP V>

7. Create a simple web service and write any distributed application to

App.py

```python
from flask import Flask, jsonify, request

app = Flask(__name__)

@app.route('/add', methods=['POST'])
def add():
    data = request.get_json()
    a = data['a']
    b = data['b']
    result = a + b
    return jsonify({'result': result})

if __name__ == '__main__':
    app.run(debug=True)
```

output



Client.py

```python
import requests

data = {
    'a': 10,
    'b': 20
}

response = requests.post('http://localhost:5000/add', json=data)
result = response.json()
print("Result of addition:", result['result'])
```

Output