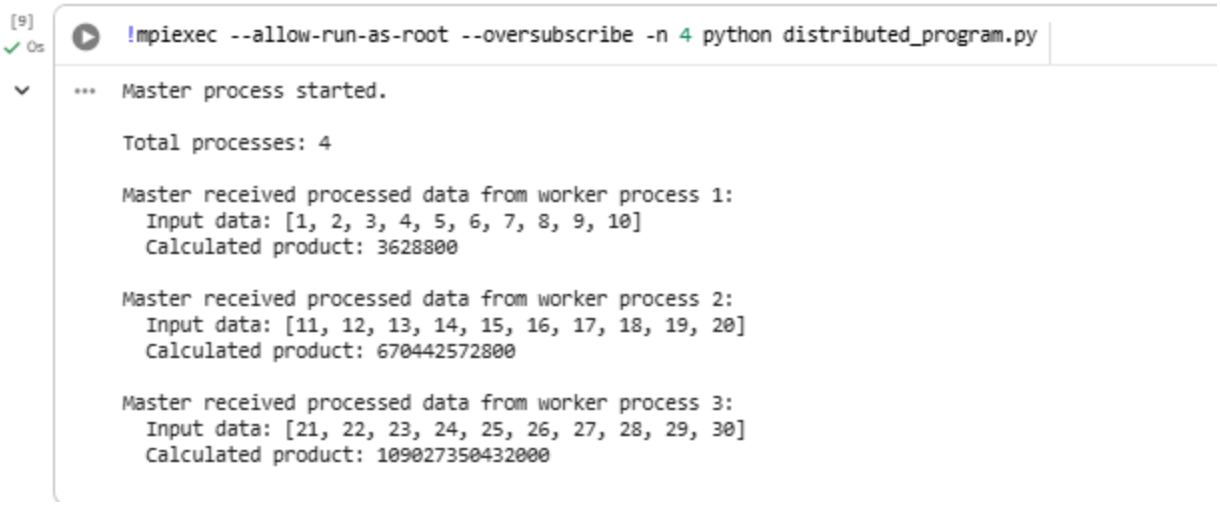
France Raphael R. Rivera

Screenshots:

```python
from mpi4py import MPI

# Initialize MPI communicator
comm = MPI.COMM_WORLD
rank = comm.Get_rank()   # Unique ID of each process
size = comm.Get_size()   # Total number of processes

# ---------------- MASTER PROCESS ----------------
if rank == 0:
    print("Master process started.\n")
    print(f"Total processes: {size}\n")

    # Receive results from all worker processes
    for worker_rank in range(1, size):
        # Receive message from the current worker
        data = comm.recv(source=worker_rank)
        print(f"Master received processed data from worker process {data['rank']}:")
        print(f"  Input data: {data['input_values']}")
        print(f"  Calculated product: {data['product']}\n")

# ---------------- WORKER PROCESSES ----------------
else:
    # Each worker gets a unique list of 10 numbers based on its rank
    start = (rank - 1) * 10 + 1
    end = start + 10   # exclusive
    numbers = list(range(start, end))

    # Compute the product of the numbers
    total_product = 1
    for num in numbers:
        total_product *= num

    # Prepare a message containing rank, task, and result
    message = {
        "rank": rank,
        "input_values": numbers,
        "product": total_product
    }

    # Send message to master process (rank 0)
    comm.send(message, dest=0)

    # Worker-side print removed for cleaner output
```

Run the MPI Program



```
[9]
✓ 0s
!mpiexec --allow-run-as-root --oversubscribe -n 4 python distributed_program.py

Master process started.

Total processes: 4

Master received processed data from worker process 1:
  Input data: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  Calculated product: 3628800

Master received processed data from worker process 2:
  Input data: [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
  Calculated product: 670442572800

Master received processed data from worker process 3:
  Input data: [21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
  Calculated product: 109027350432000
```

PART III – DISCUSSION & REFLECTION

1. Why is message passing required in distributed systems?

- In distributed systems, processes run on different nodes or separate parts of a system where they do not share the same RAM. Since one process cannot directly read the variables or data held by another, they need a formal way to communicate. Message passing acts like a postal service for data, allowing a worker process to package its results and send them to the master process. Without this, there would be no way to combine the work done by different processors into a single final result.

2. What happens if one process fails?

- In a standard MPI setup, the system is quite fragile when it comes to errors. If a worker process crashes or exits early, it fails to send the expected data back to the master. Because the master process uses a blocking receive command, it will sit and wait forever for that specific message to arrive, causing the entire program to hang or freeze. This demonstrates that basic message passing doesn't have built-in safety nets or "self-healing" features, so if one part fails, the whole job usually fails too.

3. How does this model differ from shared-memory programming?

- The main difference is in how the processes "talk" to each other and handle data. In shared-memory programming, all parts of the program can access the same memory

space directly, which is fast but can lead to messy data conflicts. In the message-passing model, every process is isolated with its own private memory. If they need to share something, they must explicitly use "send" and "receive" functions. This makes the code a bit more organized and safer from accidental data overwrites, but it requires more careful planning to move data around.