ASSIGNMENT 6
28/03/25

NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS8D
TOPIC : FORMAL METHOD
CODE : CS-18201

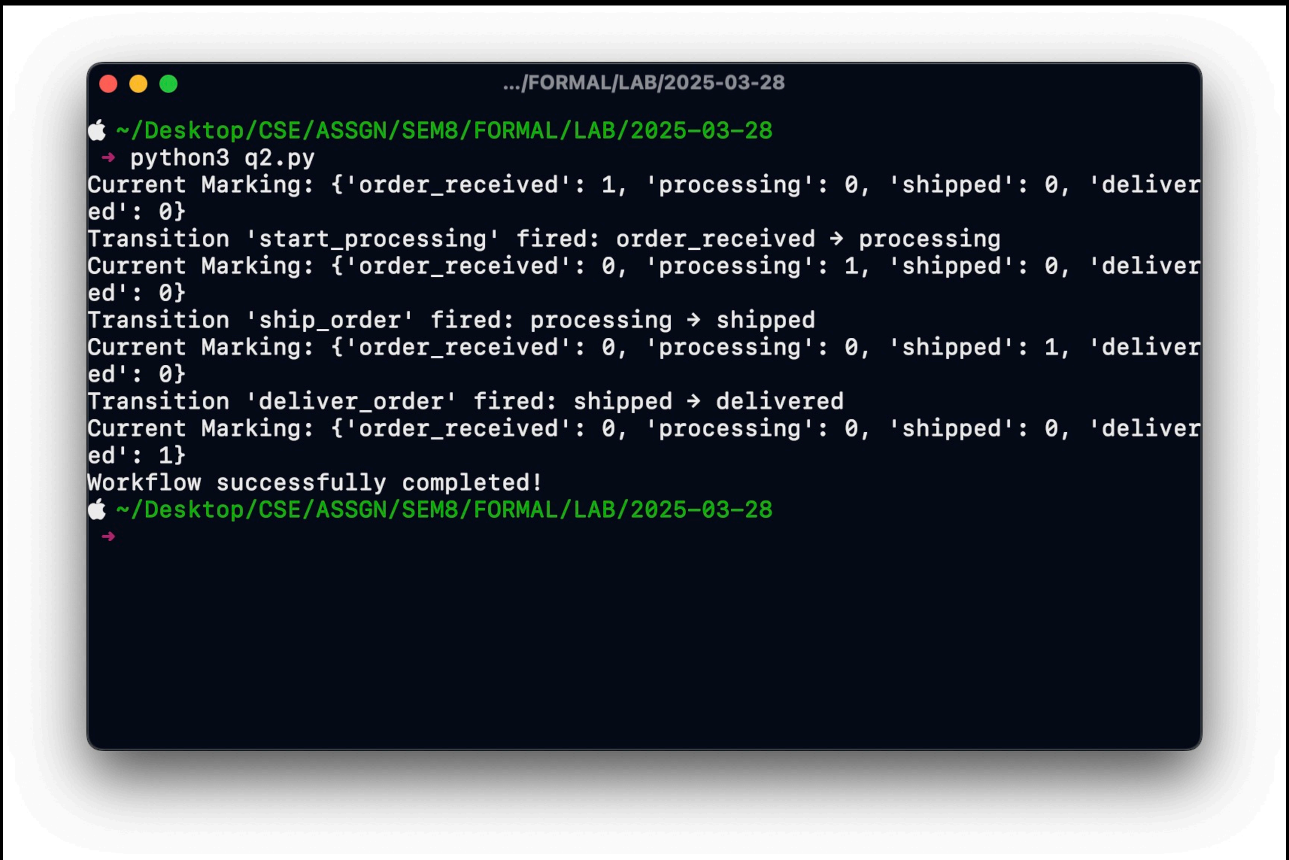
```
# Q1 Model a synchronization problem using Petri Nets
and verify deadlock freedom.

class PetriNet:
    def __init__(self):
        self.places = {
            "P1_waiting": 1,
            "P2_waiting": 1,
            "Resource_free": 1,
            "P1_in_CS": 0,
            "P2_in_CS": 0,
        }

        self.transitions = {
            "P1_enters": {"inputs": ["P1_waiting",
"Resource_free"], "outputs": ["P1_in_CS"]},
            "P1_exits": {"inputs": ["P1_in_CS"],
"outputs": ["Resource_free", "P1_waiting"]},
            "P2_enters": {"inputs": ["P2_waiting",
"Resource_free"], "outputs": ["P2_in_CS"]},
            "P2_exits": {"inputs": ["P2_in_CS"],
"outputs": ["Resource_free", "P2_waiting"]},
        }

    def can_fire(self, transition):
        for place in self.transitions[transition]
["inputs"]:
            if self.places[place] == 0:
                return False
        return True

    def fire(self, transition):
        if not self.can_fire(transition):
            print(f"Transition {transition} cannot
fire.")
            return False

        for place in self.transitions[transition]
["inputs"]:
            self.places[place] -= 1
        for place in self.transitions[transition]
["outputs"]:
            self.places[place] += 1
        print(f"Fired transition: {transition}")
        return True
```

```python
    def is_deadlock(self):
        return not any(self.can_fire(t) for t in
self.transitions)

    def display_state(self):
        print("Current State:", self.places)


pn = PetriNet()
pn.display_state()
transitions = ["P1_enters", "P1_exits", "P2_enters",
"P2_exits"]

for _ in range(5):
    for t in transitions:
        if pn.fire(t):
            pn.display_state()
    if pn.is_deadlock():
        print("Deadlock detected!")
        break
else:
    print("No deadlock, system is safe.")
```

```
.../FORMAL/LAB/2025-03-28
~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
➜ python3 q1.py
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P1_enters
Current State: {'P1_waiting': 0, 'P2_waiting': 1, 'Resource_free': 0, 'P1_in_CS': 1, 'P2_in_CS': 0}
Fired transition: P1_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P2_enters
Current State: {'P1_waiting': 1, 'P2_waiting': 0, 'Resource_free': 0, 'P1_in_CS': 0, 'P2_in_CS': 1}
Fired transition: P2_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P1_enters
Current State: {'P1_waiting': 0, 'P2_waiting': 1, 'Resource_free': 0, 'P1_in_CS': 1, 'P2_in_CS': 0}
Fired transition: P1_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P2_enters
Current State: {'P1_waiting': 1, 'P2_waiting': 0, 'Resource_free': 0, 'P1_in_CS': 0, 'P2_in_CS': 1}
Fired transition: P2_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P1_enters
Current State: {'P1_waiting': 0, 'P2_waiting': 1, 'Resource_free': 0, 'P1_in_CS': 1, 'P2_in_CS': 0}
Fired transition: P1_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P2_enters
Current State: {'P1_waiting': 1, 'P2_waiting': 0, 'Resource_free': 0, 'P1_in_CS': 0, 'P2_in_CS': 1}
Fired transition: P2_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P1_enters
Current State: {'P1_waiting': 0, 'P2_waiting': 1, 'Resource_free': 0, 'P1_in_CS': 1, 'P2_in_CS': 0}
Fired transition: P1_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P2_enters
Current State: {'P1_waiting': 1, 'P2_waiting': 0, 'Resource_free': 0, 'P1_in_CS': 0, 'P2_in_CS': 1}
Fired transition: P2_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P1_enters
Current State: {'P1_waiting': 0, 'P2_waiting': 1, 'Resource_free': 0, 'P1_in_CS': 1, 'P2_in_CS': 0}
Fired transition: P1_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
Fired transition: P2_enters
Current State: {'P1_waiting': 1, 'P2_waiting': 0, 'Resource_free': 0, 'P1_in_CS': 0, 'P2_in_CS': 1}
Fired transition: P2_exits
Current State: {'P1_waiting': 1, 'P2_waiting': 1, 'Resource_free': 1, 'P1_in_CS': 0, 'P2_in_CS': 0}
No deadlock, system is safe.
~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
➜
```

```python
# Q2 Implement a basic workflow system (e.g., an order
processing system) using Petri Nets and
# analyze its correctness.

class PetriNet:
    def __init__(self):
        self.places = {"order_received": 1,
"processing": 0, "shipped": 0, "delivered": 0}
        self.transitions = {
            "start_processing": {"input":
"order_received", "output": "processing"},
            "ship_order": {"input": "processing",
"output": "shipped"},
            "deliver_order": {"input": "shipped",
"output": "delivered"}
        }

    def fire_transition(self, transition_name):
        if transition_name in self.transitions:
            transition =
self.transitions[transition_name]
            input_place = transition["input"]
            output_place = transition["output"]

            if self.places[input_place] > 0:
                self.places[input_place] -= 1
                self.places[output_place] += 1
                print(f"Transition '{transition_name}'
fired: {input_place} → {output_place}")
            else:
                print(f"Cannot fire
'{transition_name}': No tokens in {input_place}")
        else:
            print(f"Transition '{transition_name}' not
found.")

    def is_reachable(self, target_state):
        return all(self.places[p] == target_state[p]
for p in self.places)

    def print_state(self):
        print("Current Marking:", self.places)

net = PetriNet()
net.print_state()
```

```python
net.fire_transition("start_processing")
net.print_state()
net.fire_transition("ship_order")
net.print_state()
net.fire_transition("deliver_order")
net.print_state()

final_state = {"order_received": 0, "processing": 0,
"shipped": 0, "delivered": 1}
if net.is_reachable(final_state):
    print("Workflow successfully completed!")
else:
    print("Workflow did not reach final state.")
```



```
.../FORMAL/LAB/2025-03-28
~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
→ python3 q2.py
Current Marking: {'order_received': 1, 'processing': 0, 'shipped': 0, 'deliver
ed': 0}
Transition 'start_processing' fired: order_received → processing
Current Marking: {'order_received': 0, 'processing': 1, 'shipped': 0, 'deliver
ed': 0}
Transition 'ship_order' fired: processing → shipped
Current Marking: {'order_received': 0, 'processing': 0, 'shipped': 1, 'deliver
ed': 0}
Transition 'deliver_order' fired: shipped → delivered
Current Marking: {'order_received': 0, 'processing': 0, 'shipped': 0, 'deliver
ed': 1}
Workflow successfully completed!
~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
→
```

```python
# Q3 Simulate a client-server communication model using
CCS (Calculus of Communicating
# Systems).

## server.py
import socket

def main():
```

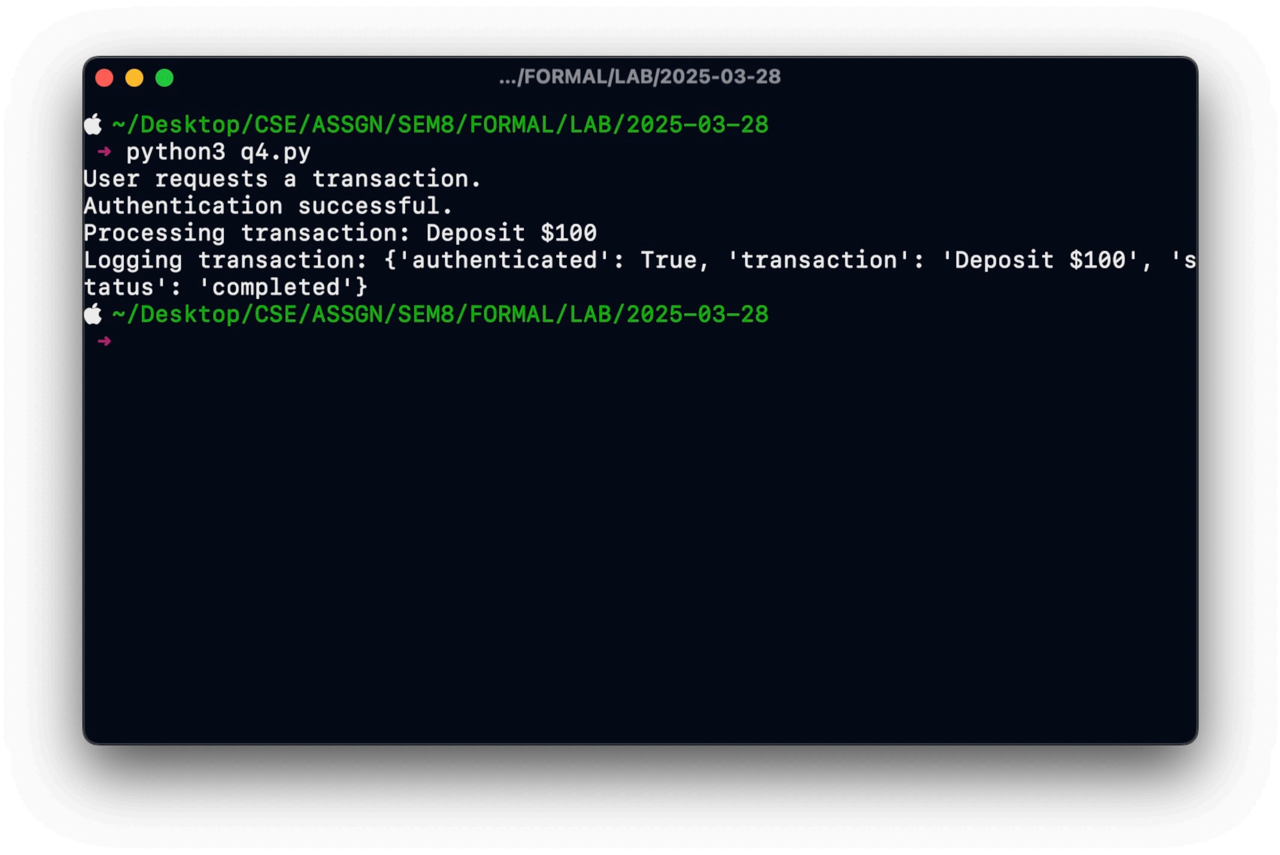
```python
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind(("localhost", 12345))
    server_socket.listen(1)

    print("Server is listening...")
    conn, addr = server_socket.accept()
    print(f"Connection established with {addr}")

    message = conn.recv(1024).decode()
    print(f"Server received: {message}")

    response = "Hello, Client!"
    print(f"Server sends: {response}")
    conn.sendall(response.encode())

    conn.close()
    server_socket.close()


if __name__ == "__main__":
    main()

# Q3 Simulate a client-server communication model using
CCS (Calculus of Communicating
# Systems).

## client.py
import socket

def main():
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect(("localhost", 12345))
    message = "Hello, Server!"
    print(f"Client sends: {message}")
    client_socket.sendall(message.encode())
    response = client_socket.recv(1024).decode()
    print(f"Client received: {response}")

    client_socket.close()

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

```
.../FORMAL/LAB/2025-03-28

~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
➜ python3 q3s.py
Server is listening...
Connection established with ('127.0.0.1', 50740)
Server received: Hello, Server!
Server sends: Hello, Client!
~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
➜
```

```
~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
➜ python3 q3c.py
Client sends: Hello, Server!
Client received: Hello, Client!
~/Desktop/CSE/ASSGN/SEM8/FORMAL/LAB/2025-03-28
➜
```

```python
# Q4 Develop a formal specification of an online
transaction processing system using Process Algebra.

class Process:
    def execute(self, input_data):
        raise NotImplementedError("Subclasses must
implement the execute method.")

class User(Process):
    def execute(self, input_data):
        print("User requests a transaction.")
```

```python
        return ("authenticate", input_data)

class Authentication(Process):
    def execute(self, input_data):
        if input_data.get("authenticated", False):
            print("Authentication successful.")
            return ("process_transaction", input_data)
        else:
            print("Authentication failed.")
            return ("terminate", None)

class Transaction(Process):
    def execute(self, input_data):
        print(f"Processing transaction:
{input_data['transaction']}")
        input_data["status"] = "completed"
        return ("log_transaction", input_data)

class Logger(Process):
    def execute(self, input_data):
        print(f"Logging transaction: {input_data}")
        return ("terminate", None)

class OLTPSystem:
    def __init__(self):
        self.processes = {
            "user": User(),
            "authenticate": Authentication(),
            "process_transaction": Transaction(),
            "log_transaction": Logger()
        }
    def run(self, input_data):
        current_process = "user"
        while current_process != "terminate":
            next_process, input_data =
self.processes[current_process].execute(input_data)
            current_process = next_process
if __name__ == "__main__":
    input_data = {
        "authenticated": True,
        "transaction": "Deposit $100"
    }
    system = OLTPSystem()
    system.run(input_data)
```




```python
# Q5 Implement a distributed computation model using
Pi-Calculus for mobile process interactions.

import threading
import queue

class Channel:
    def __init__(self):
        self._queue = queue.Queue()

    def send(self, message):
        self._queue.put(message)

    def receive(self):
        return self._queue.get()

class Process(threading.Thread):
    def __init__(self, name, channel, action):
        super().__init__()
        self.name = name
        self.channel = channel
        self.action = action

    def run(self):
        self.action(self.channel)

def action_a(channel):
```

```python
    print("Process A: Waiting to receive a message.")
    message = channel.receive()
    print(f"Process A: Received message '{message}'")
    print("Process A: Sending response.")
    channel.send("Hello from Process A")

def action_b(channel):
    print("Process B: Sending message to Process A.")
    channel.send("Hello from Process B")
    message = channel.receive()
    print(f"Process B: Received message '{message}'")

channel = Channel()

process_a = Process("Process A", channel, action_a)
process_b = Process("Process B", channel, action_b)

process_b.start()
process_a.start()

process_b.join()
process_a.join()

print("Both processes have completed their
interactions.")
```