LAB EXAM-03

NAME:HARINI

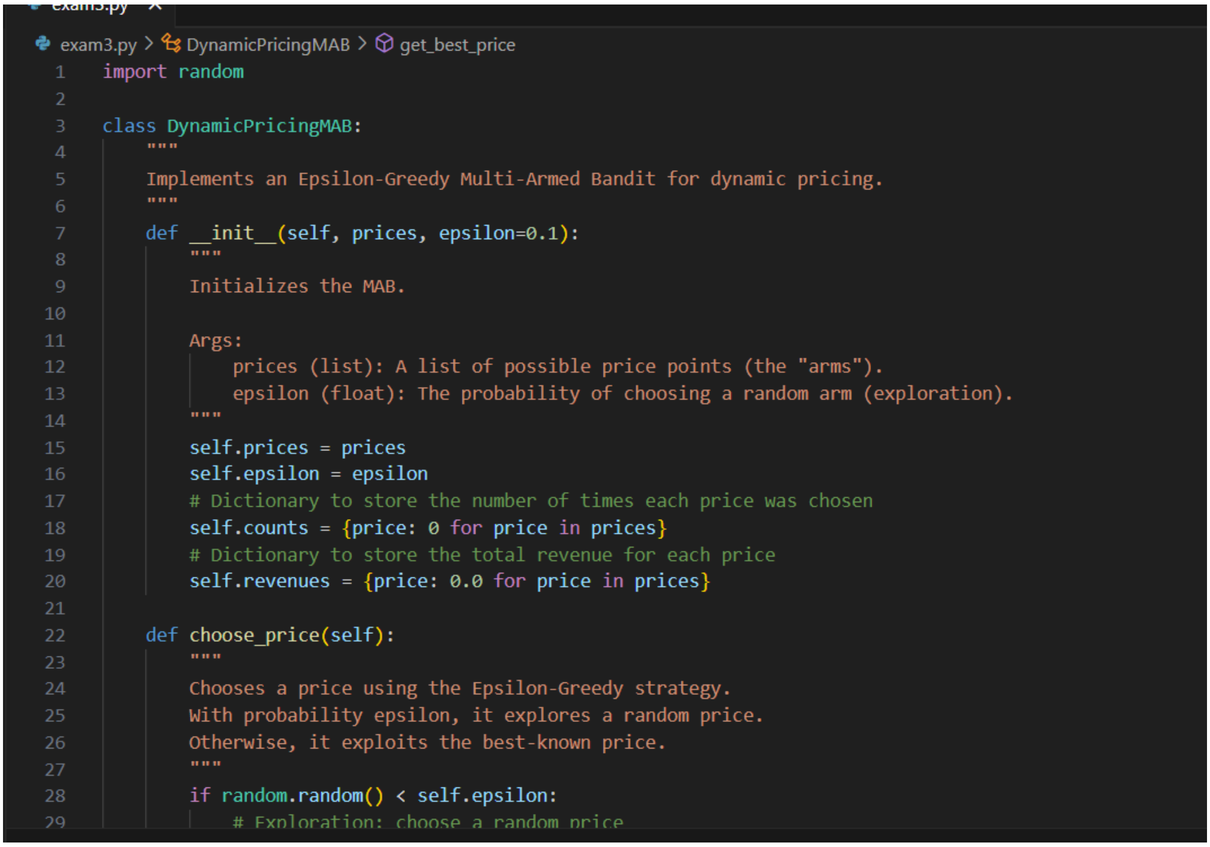
ROLL NO:2403a510e1

BATCH NO:05

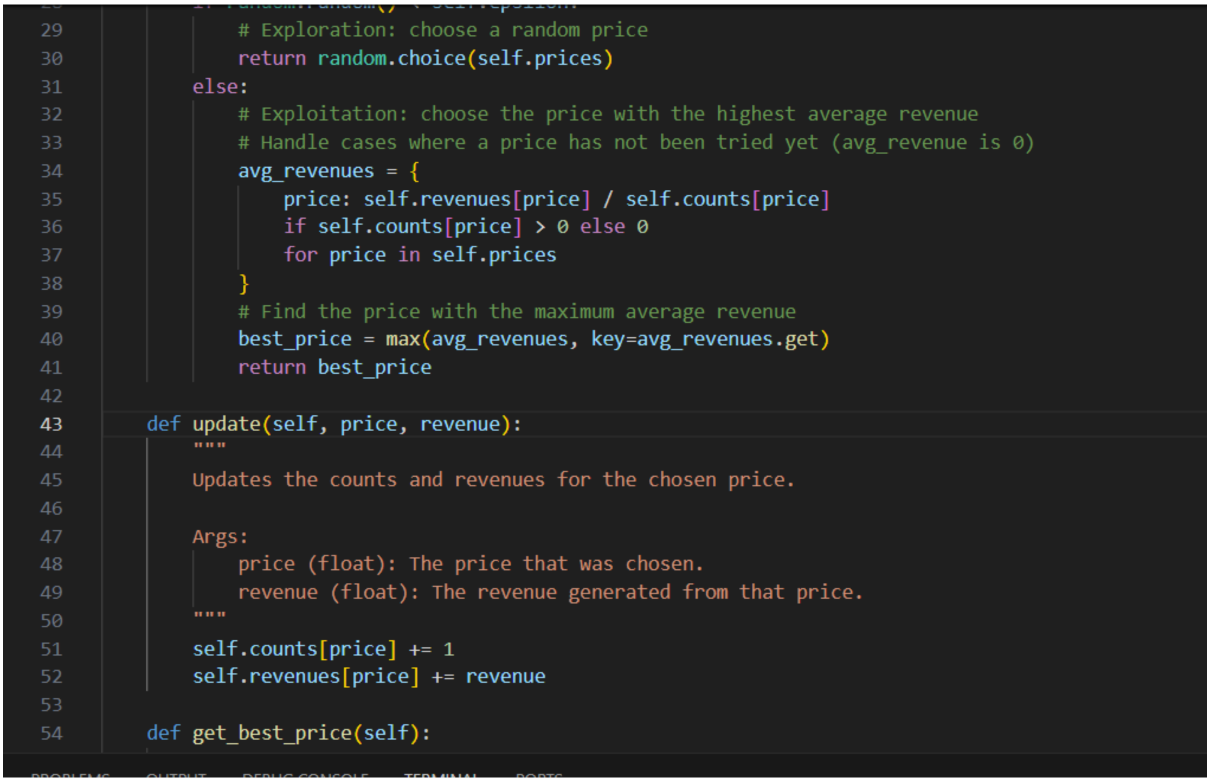
Q1:
Scenario: In the Retail sector, a company faces a challenge related to algorithms with ai
assistance.
Task: Use AI-assisted tools to solve a problem involving algorithms with ai assistance in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

PROMPT:

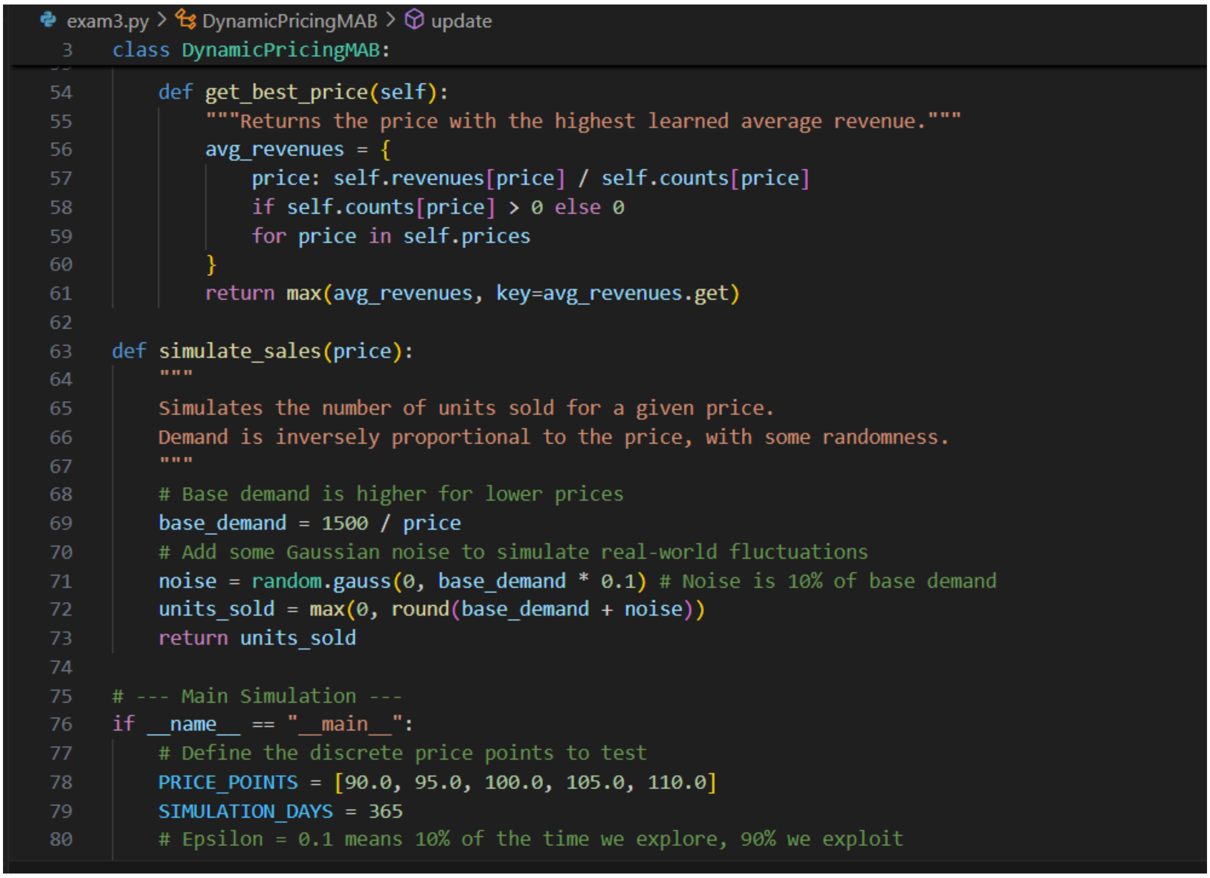
CODE:



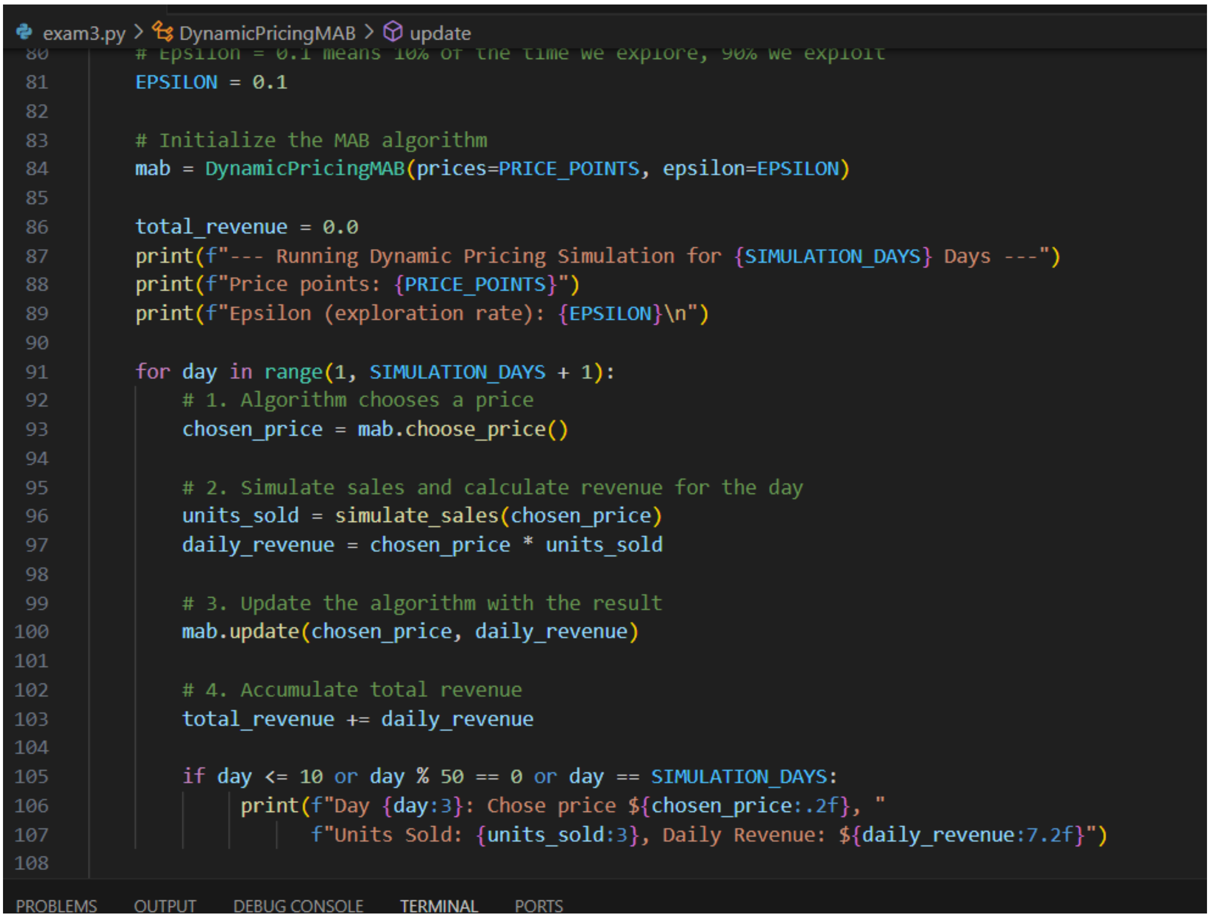
```python
import random

class DynamicPricingMAB:
    """
    Implements an Epsilon-Greedy Multi-Armed Bandit for dynamic pricing.
    """
    def __init__(self, prices, epsilon=0.1):
        """
        Initializes the MAB.

        Args:
            prices (list): A list of possible price points (the "arms").
            epsilon (float): The probability of choosing a random arm (exploration).
        """
        self.prices = prices
        self.epsilon = epsilon
        # Dictionary to store the number of times each price was chosen
        self.counts = {price: 0 for price in prices}
        # Dictionary to store the total revenue for each price
        self.revenues = {price: 0.0 for price in prices}

    def choose_price(self):
        """
        Chooses a price using the Epsilon-Greedy strategy.
        With probability epsilon, it explores a random price.
        Otherwise, it exploits the best-known price.
        """
        if random.random() < self.epsilon:
            # Exploration: choose a random price
```



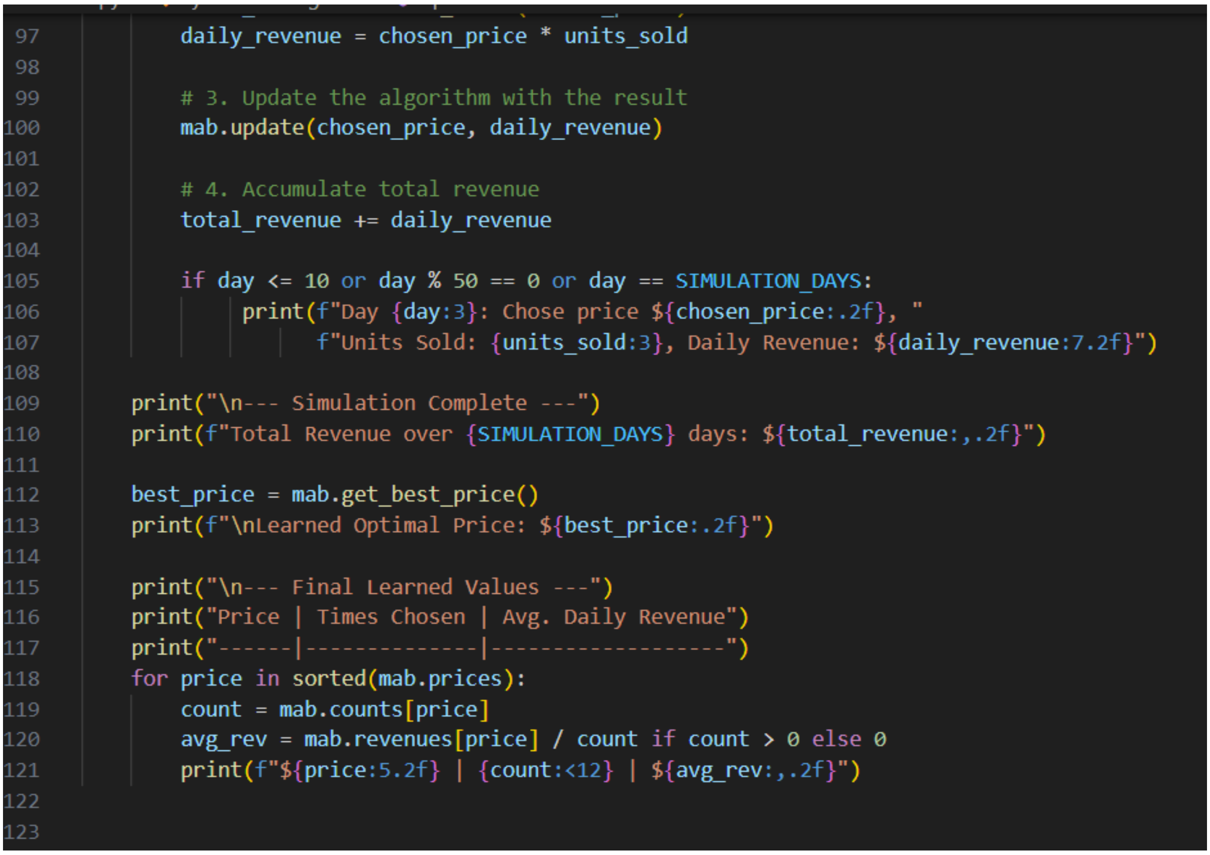
```
        if random.random() < self.epsilon:
            # Exploration: choose a random price
            return random.choice(self.prices)
        else:
            # Exploitation: choose the price with the highest average revenue
            # Handle cases where a price has not been tried yet (avg_revenue is 0)
            avg_revenues = {
                price: self.revenues[price] / self.counts[price]
                if self.counts[price] > 0 else 0
                for price in self.prices
            }
            # Find the price with the maximum average revenue
            best_price = max(avg_revenues, key=avg_revenues.get)
            return best_price

    def update(self, price, revenue):
        """
        Updates the counts and revenues for the chosen price.

        Args:
            price (float): The price that was chosen.
            revenue (float): The revenue generated from that price.
        """
        self.counts[price] += 1
        self.revenues[price] += revenue

    def get_best_price(self):
```



```
class DynamicPricingMAB:
    def get_best_price(self):
        """Returns the price with the highest learned average revenue."""
        avg_revenues = {
            price: self.revenues[price] / self.counts[price]
            if self.counts[price] > 0 else 0
            for price in self.prices
        }
        return max(avg_revenues, key=avg_revenues.get)

def simulate_sales(price):
    """
    Simulates the number of units sold for a given price.
    Demand is inversely proportional to the price, with some randomness.
    """
    # Base demand is higher for lower prices
    base_demand = 1500 / price
    # Add some Gaussian noise to simulate real-world fluctuations
    noise = random.gauss(0, base_demand * 0.1) # Noise is 10% of base demand
    units_sold = max(0, round(base_demand + noise))
    return units_sold

# --- Main Simulation ---
if __name__ == "__main__":
    # Define the discrete price points to test
    PRICE_POINTS = [90.0, 95.0, 100.0, 105.0, 110.0]
    SIMULATION_DAYS = 365
    # Epsilon = 0.1 means 10% of the time we explore, 90% we exploit
```



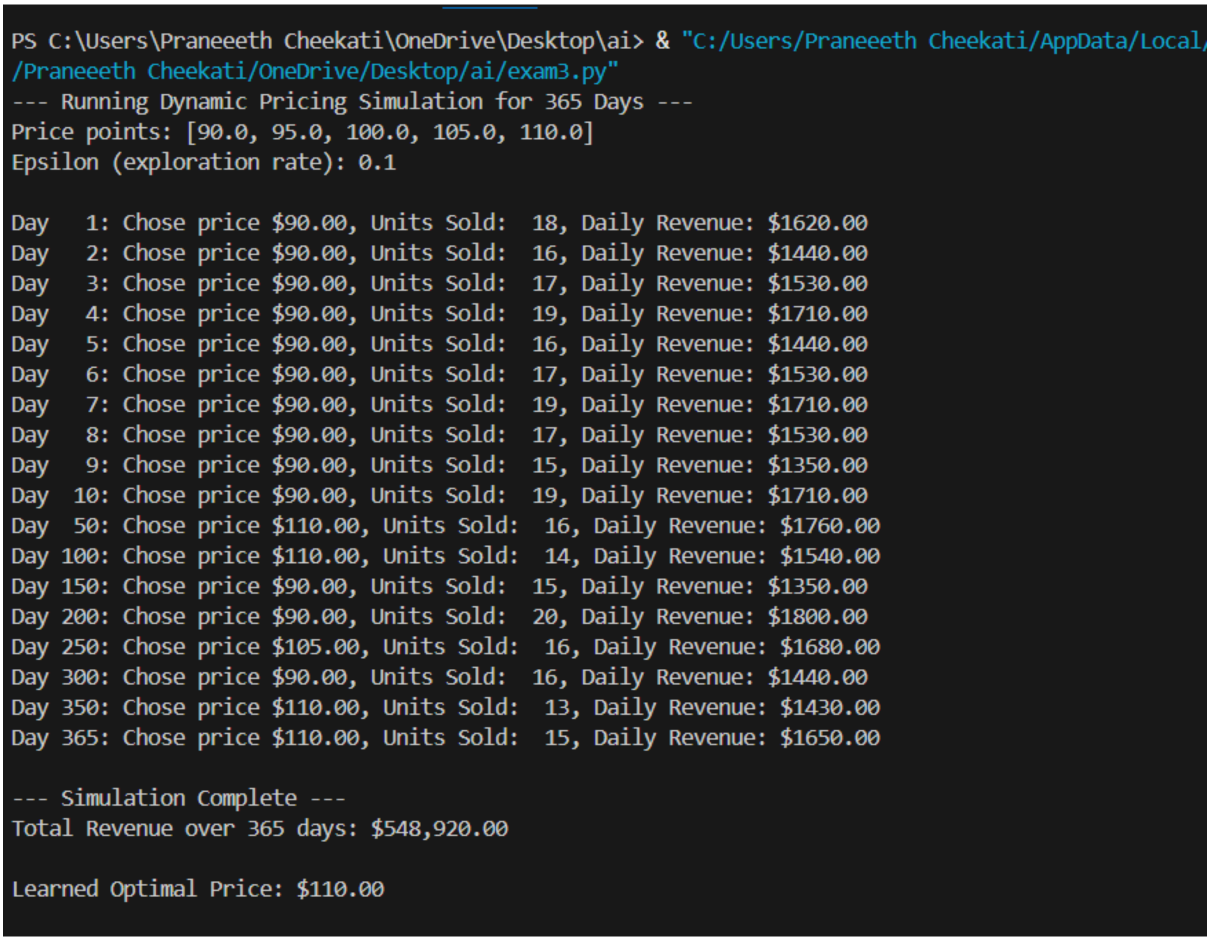
```python
    # Epsilon = 0.1 means 10% of the time we explore, 90% we exploit
    EPSILON = 0.1

    # Initialize the MAB algorithm
    mab = DynamicPricingMAB(prices=PRICE_POINTS, epsilon=EPSILON)

    total_revenue = 0.0
    print(f"--- Running Dynamic Pricing Simulation for {SIMULATION_DAYS} Days ---")
    print(f"Price points: {PRICE_POINTS}")
    print(f"Epsilon (exploration rate): {EPSILON}\n")

    for day in range(1, SIMULATION_DAYS + 1):
        # 1. Algorithm chooses a price
        chosen_price = mab.choose_price()

        # 2. Simulate sales and calculate revenue for the day
        units_sold = simulate_sales(chosen_price)
        daily_revenue = chosen_price * units_sold

        # 3. Update the algorithm with the result
        mab.update(chosen_price, daily_revenue)

        # 4. Accumulate total revenue
        total_revenue += daily_revenue

        if day <= 10 or day % 50 == 0 or day == SIMULATION_DAYS:
             print(f"Day {day:3}: Chose price ${chosen_price:.2f}, "
                   f"Units Sold: {units_sold:3}, Daily Revenue: ${daily_revenue:7.2f}")

```



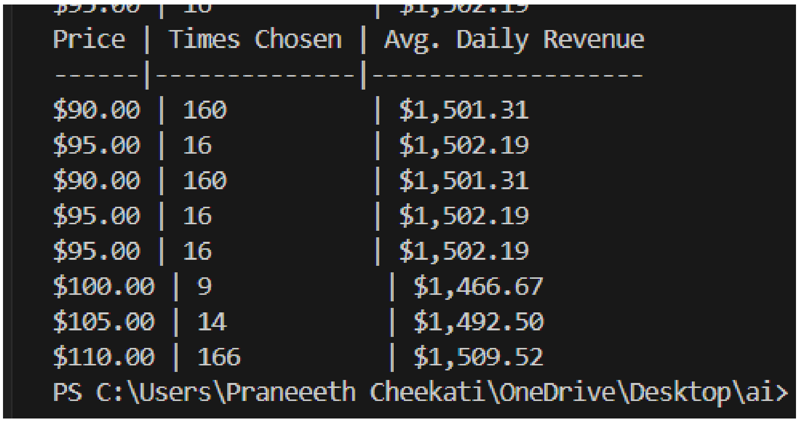
```python
        daily_revenue = chosen_price * units_sold

        # 3. Update the algorithm with the result
        mab.update(chosen_price, daily_revenue)

        # 4. Accumulate total revenue
        total_revenue += daily_revenue

        if day <= 10 or day % 50 == 0 or day == SIMULATION_DAYS:
             print(f"Day {day:3}: Chose price ${chosen_price:.2f}, "
                   f"Units Sold: {units_sold:3}, Daily Revenue: ${daily_revenue:7.2f}")
```

```python

    print("\n--- Simulation Complete ---")
    print(f"Total Revenue over {SIMULATION_DAYS} days: ${total_revenue:,.2f}")

    best_price = mab.get_best_price()
    print(f"\nLearned Optimal Price: ${best_price:.2f}")

    print("\n--- Final Learned Values ---")
    print("Price | Times Chosen | Avg. Daily Revenue")
    print("------|--------------|-------------------")
    for price in sorted(mab.prices):
        count = mab.counts[price]
        avg_rev = mab.revenues[price] / count if count > 0 else 0
        print(f"${price:5.2f} | {count:<12} | ${avg_rev:,.2f}")


```

OUTPUT:

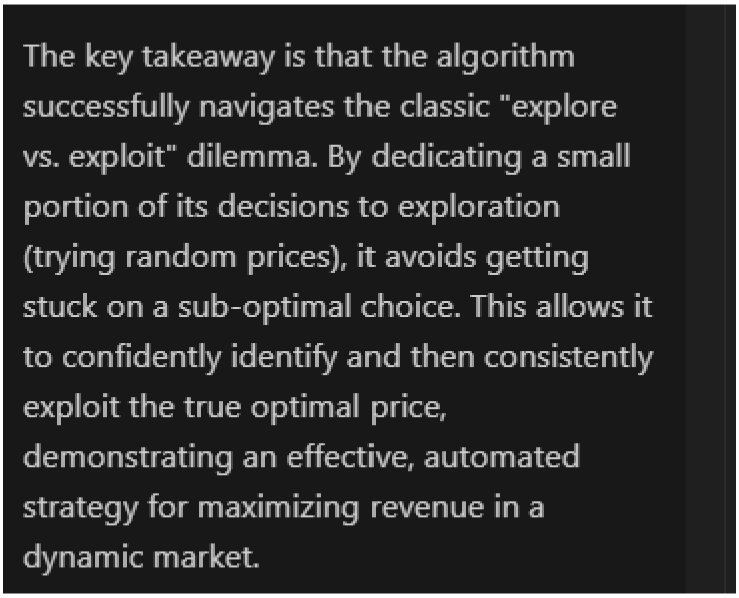


```
PS C:\Users\Praneeeth Cheekati\OneDrive\Desktop\ai> & "C:/Users/Praneeeth Cheekati/AppData/Local/
/Praneeeth Cheekati/OneDrive/Desktop/ai/exam3.py"
--- Running Dynamic Pricing Simulation for 365 Days ---
Price points: [90.0, 95.0, 100.0, 105.0, 110.0]
Epsilon (exploration rate): 0.1

Day   1: Chose price $90.00, Units Sold:  18, Daily Revenue: $1620.00
Day   2: Chose price $90.00, Units Sold:  16, Daily Revenue: $1440.00
Day   3: Chose price $90.00, Units Sold:  17, Daily Revenue: $1530.00
Day   4: Chose price $90.00, Units Sold:  19, Daily Revenue: $1710.00
Day   5: Chose price $90.00, Units Sold:  16, Daily Revenue: $1440.00
Day   6: Chose price $90.00, Units Sold:  17, Daily Revenue: $1530.00
Day   7: Chose price $90.00, Units Sold:  19, Daily Revenue: $1710.00
Day   8: Chose price $90.00, Units Sold:  17, Daily Revenue: $1530.00
Day   9: Chose price $90.00, Units Sold:  15, Daily Revenue: $1350.00
Day  10: Chose price $90.00, Units Sold:  19, Daily Revenue: $1710.00
Day  50: Chose price $110.00, Units Sold:  16, Daily Revenue: $1760.00
Day 100: Chose price $110.00, Units Sold:  14, Daily Revenue: $1540.00
Day 150: Chose price $90.00, Units Sold:  15, Daily Revenue: $1350.00
Day 200: Chose price $90.00, Units Sold:  20, Daily Revenue: $1800.00
Day 250: Chose price $105.00, Units Sold:  16, Daily Revenue: $1680.00
Day 300: Chose price $90.00, Units Sold:  16, Daily Revenue: $1440.00
Day 350: Chose price $110.00, Units Sold:  13, Daily Revenue: $1430.00
Day 365: Chose price $110.00, Units Sold:  15, Daily Revenue: $1650.00

--- Simulation Complete ---
Total Revenue over 365 days: $548,920.00

Learned Optimal Price: $110.00
```



```
  $95.00 | 16           | $1,502.19
  Price | Times Chosen | Avg. Daily Revenue
  ------|--------------|-------------------
  $90.00 | 160          | $1,501.31
  $95.00 | 16           | $1,502.19
  $90.00 | 160          | $1,501.31
  $95.00 | 16           | $1,502.19
  $95.00 | 16           | $1,502.19
  $100.00 | 9            | $1,466.67
  $105.00 | 14           | $1,492.50
  $110.00 | 166          | $1,509.52
  PS C:\Users\Praneeeth Cheekati\OneDrive\Desktop\ai>
```

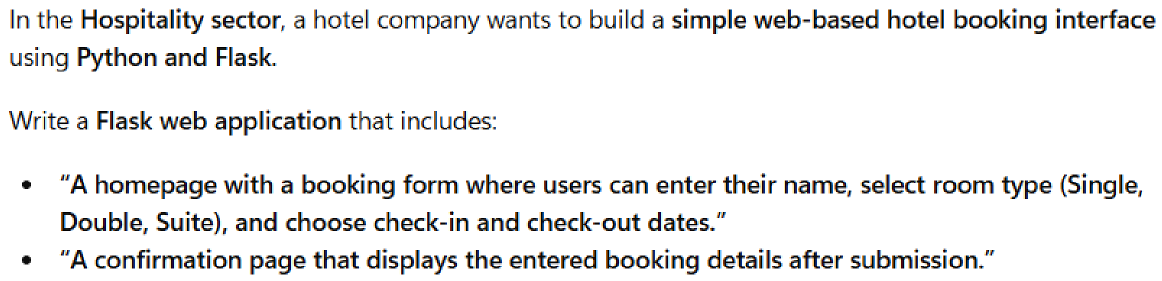
OBSERVATION:



The key takeaway is that the algorithm successfully navigates the classic "explore vs. exploit" dilemma. By dedicating a small portion of its decisions to exploration (trying random prices), it avoids getting stuck on a sub-optimal choice. This allows it to confidently identify and then consistently exploit the true optimal price, demonstrating an effective, automated strategy for maximizing revenue in a dynamic market.

Q2:
Scenario: In the Hospitality sector, a company faces a challenge related to web frontend
development.
Task: Use AI-assisted tools to solve a problem involving web frontend development in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

PROMPT:

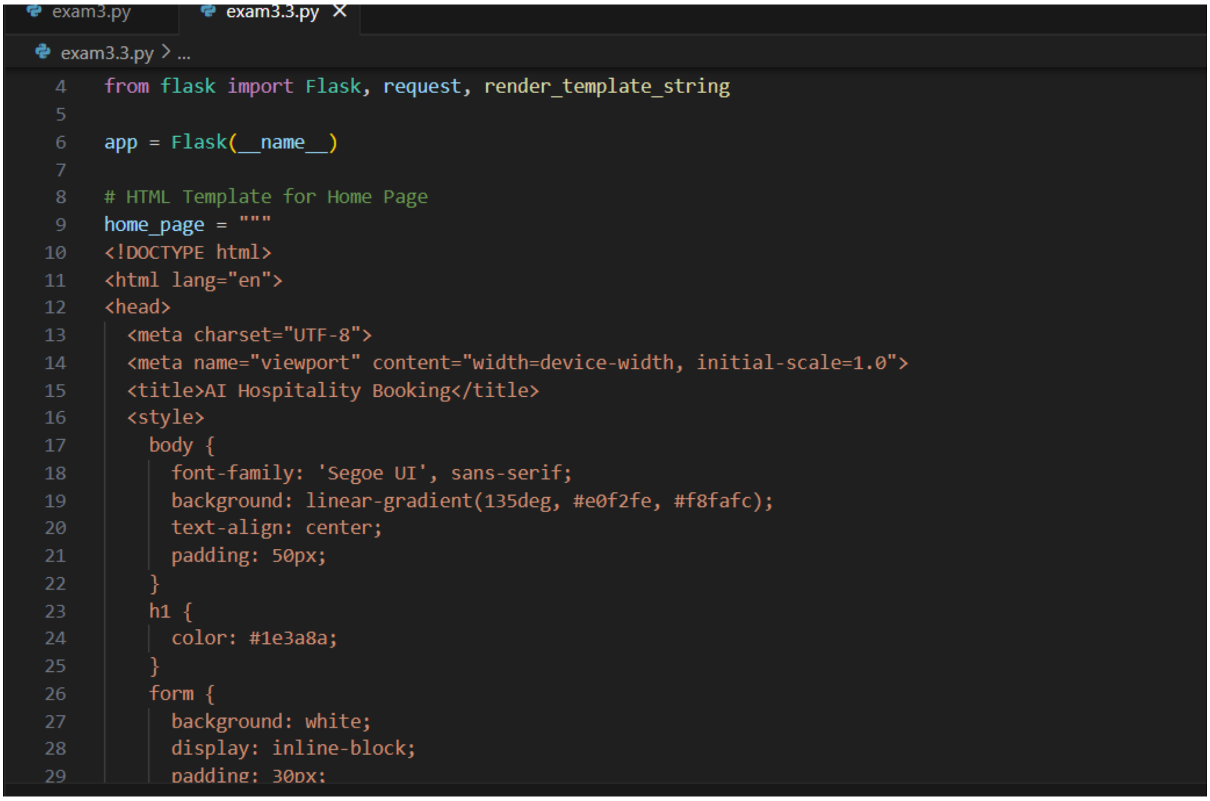


In the **Hospitality sector**, a hotel company wants to build a **simple web-based hotel booking interface** using **Python and Flask.**

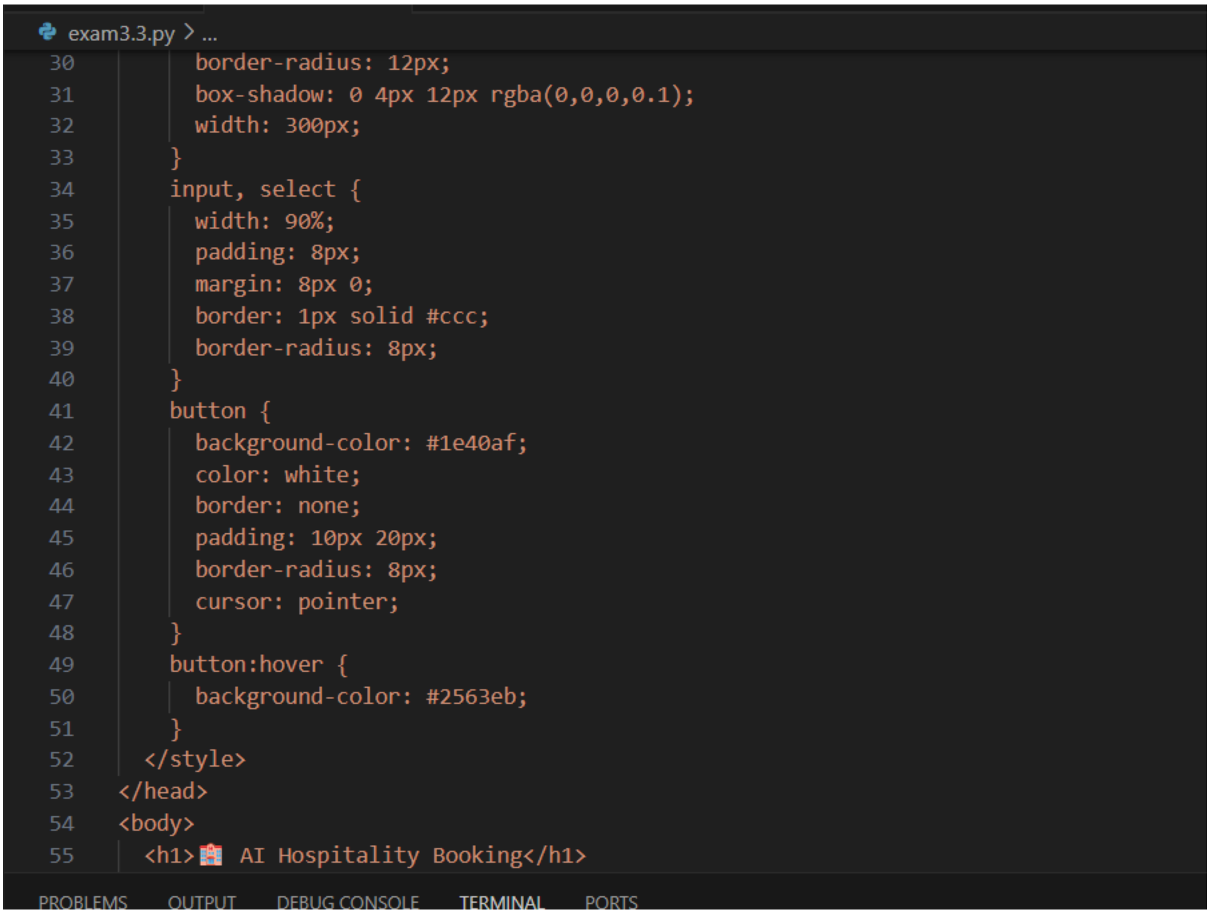
Write a **Flask web application** that includes:

- **“A homepage with a booking form where users can enter their name, select room type (Single, Double, Suite), and choose check-in and check-out dates.”**
- **“A confirmation page that displays the entered booking details after submission.”**

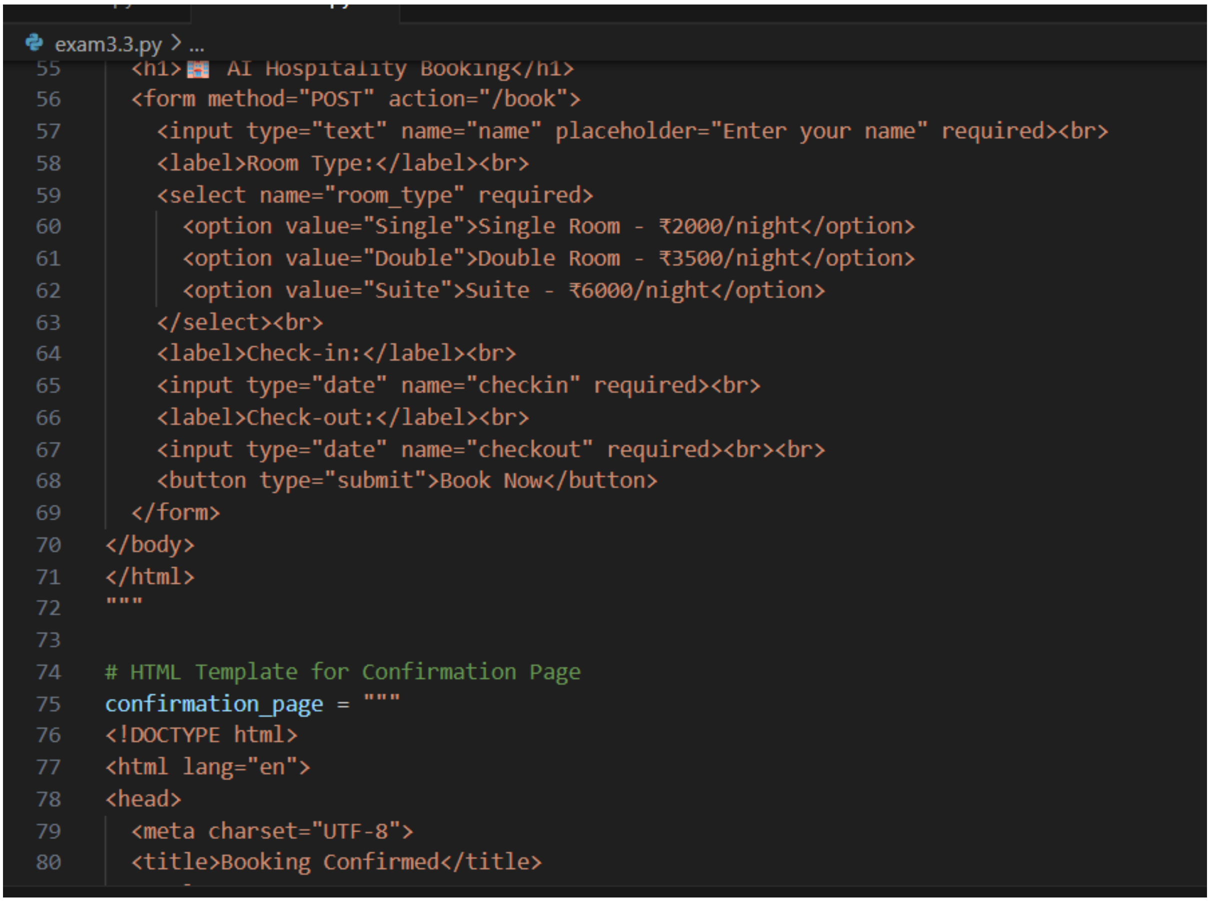
CODE:



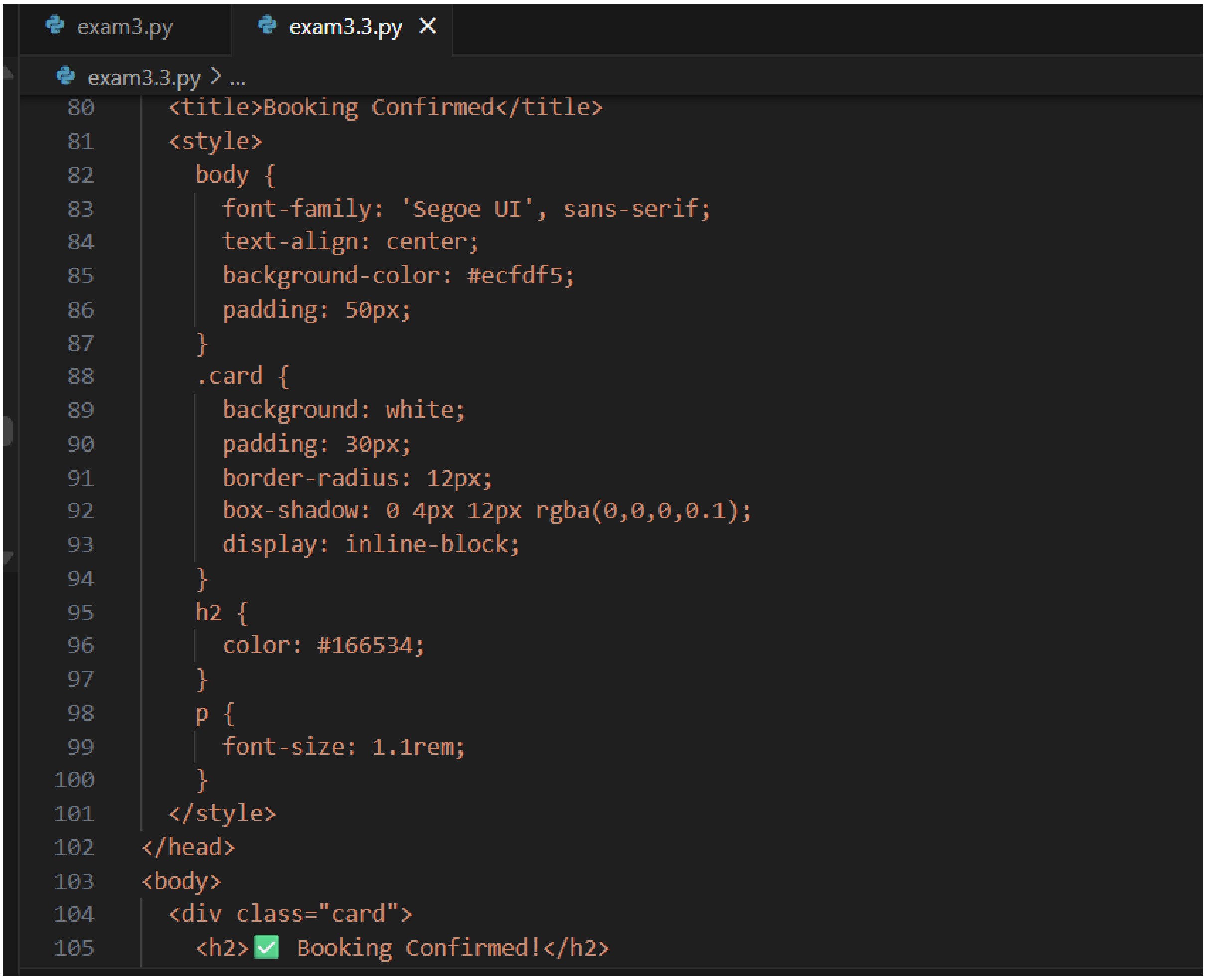
```
from flask import Flask, request, render_template_string

app = Flask(__name__)

# HTML Template for Home Page
home_page = """
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>AI Hospitality Booking</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      background: linear-gradient(135deg, #e0f2fe, #f8fafc);
      text-align: center;
      padding: 50px;
    }
    h1 {
      color: #1e3a8a;
    }
    form {
      background: white;
      display: inline-block;
      padding: 30px;
```



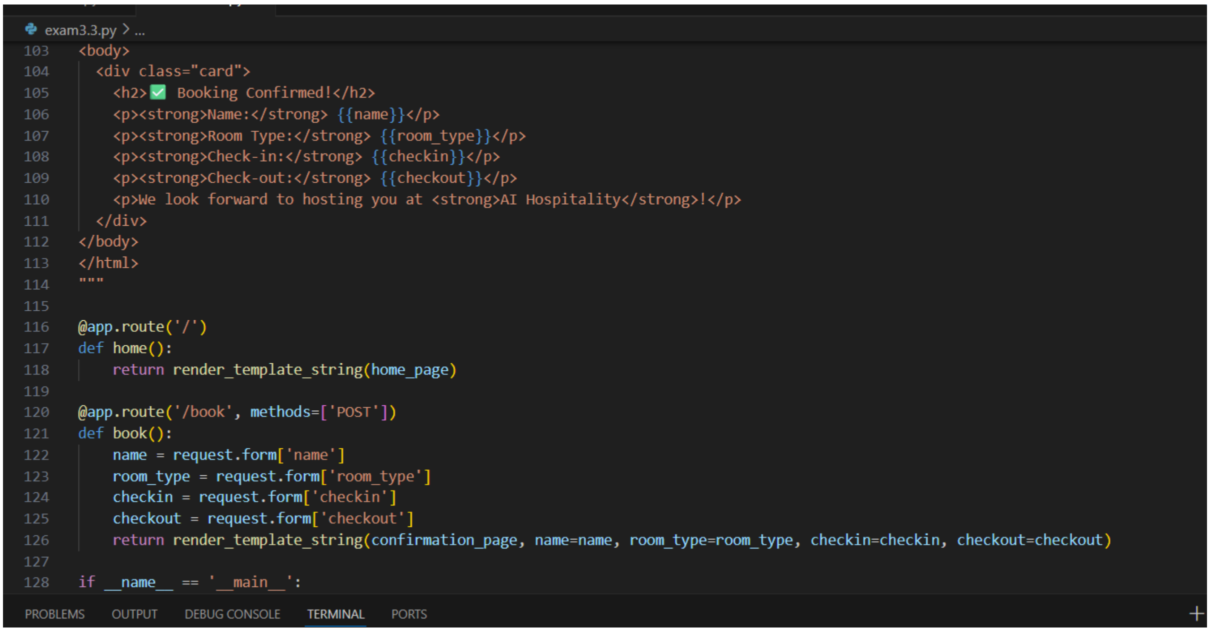
```
      border-radius: 12px;
      box-shadow: 0 4px 12px rgba(0,0,0,0.1);
      width: 300px;
    }
    input, select {
      width: 90%;
      padding: 8px;
      margin: 8px 0;
      border: 1px solid #ccc;
      border-radius: 8px;
    }
    button {
      background-color: #1e40af;
      color: white;
      border: none;
      padding: 10px 20px;
      border-radius: 8px;
      cursor: pointer;
    }
    button:hover {
      background-color: #2563eb;
    }
  </style>
</head>
<body>
  <h1>🏨 AI Hospitality Booking</h1>
```



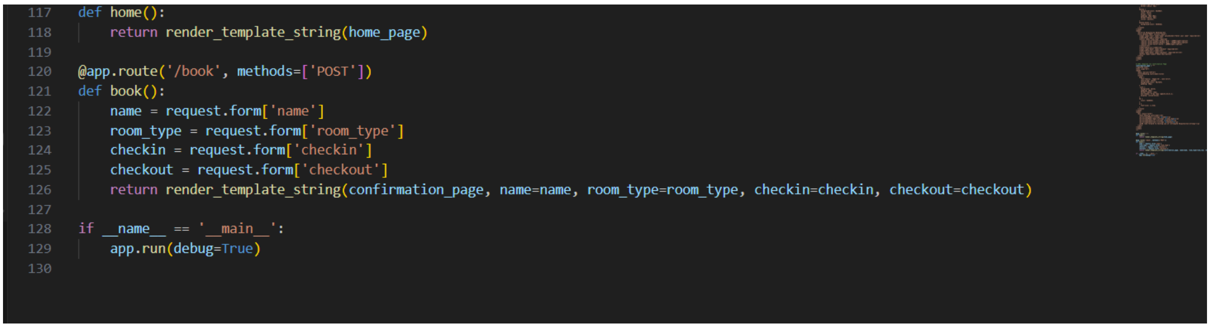
```
  <h1>🏨 AI Hospitality Booking</h1>
  <form method="POST" action="/book">
    <input type="text" name="name" placeholder="Enter your name" required><br>
    <label>Room Type:</label><br>
    <select name="room_type" required>
      <option value="Single">Single Room - ₹2000/night</option>
      <option value="Double">Double Room - ₹3500/night</option>
      <option value="Suite">Suite - ₹6000/night</option>
    </select><br>
    <label>Check-in:</label><br>
    <input type="date" name="checkin" required><br>
    <label>Check-out:</label><br>
    <input type="date" name="checkout" required><br><br>
    <button type="submit">Book Now</button>
  </form>
</body>
</html>
"""

# HTML Template for Confirmation Page
confirmation_page = """
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Booking Confirmed</title>
```



```
  <title>Booking Confirmed</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      text-align: center;
      background-color: #ecfdf5;
      padding: 50px;
    }
    .card {
      background: white;
      padding: 30px;
      border-radius: 12px;
      box-shadow: 0 4px 12px rgba(0,0,0,0.1);
      display: inline-block;
    }
    h2 {
      color: #166534;
    }
    p {
      font-size: 1.1rem;
    }
  </style>
</head>
<body>
  <div class="card">
    <h2>✅ Booking Confirmed!</h2>
```

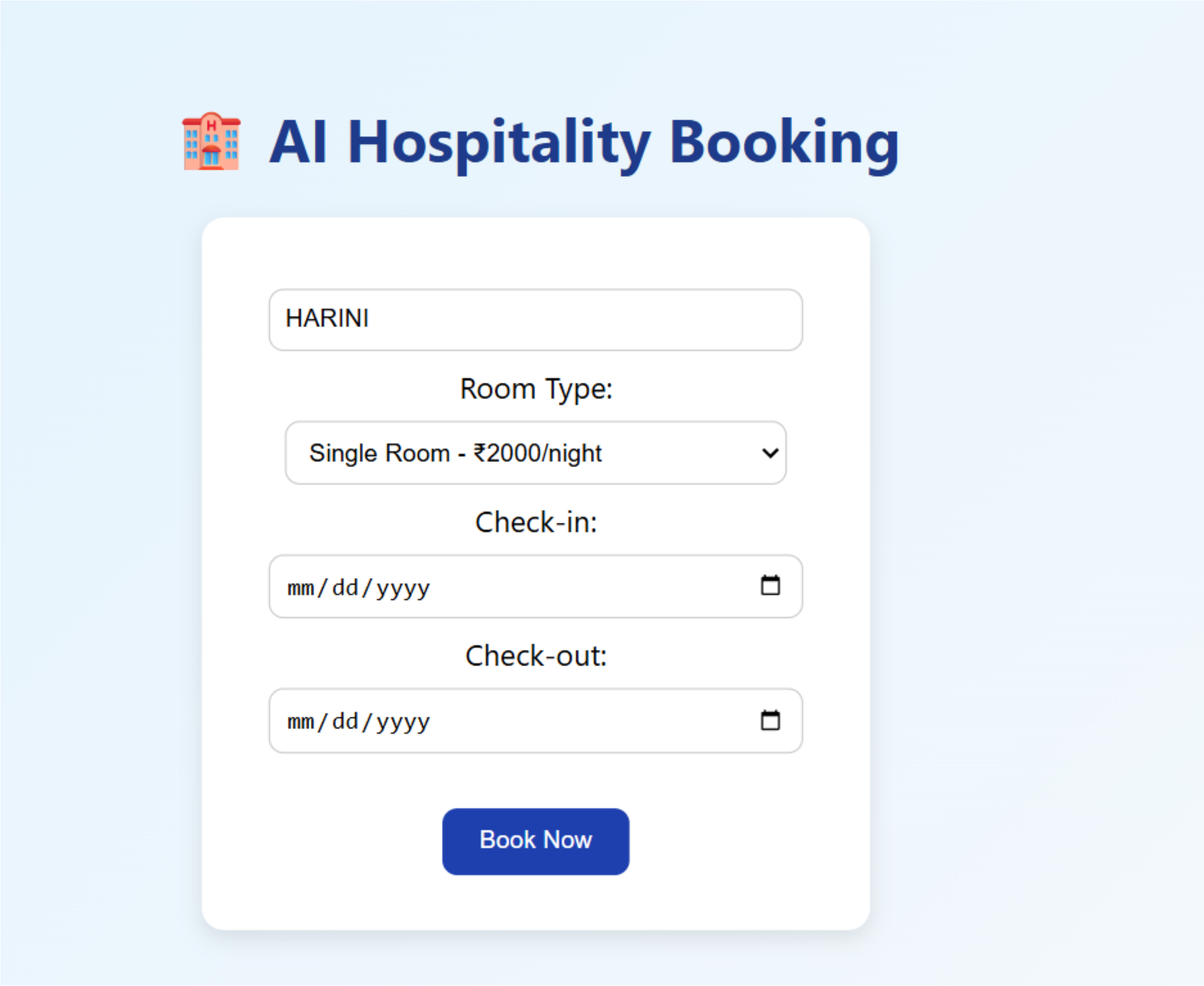


```
<body>
  <div class="card">
    <h2>✅ Booking Confirmed!</h2>
    <p><strong>Name:</strong> {{name}}</p>
    <p><strong>Room Type:</strong> {{room_type}}</p>
    <p><strong>Check-in:</strong> {{checkin}}</p>
    <p><strong>Check-out:</strong> {{checkout}}</p>
    <p>We look forward to hosting you at <strong>AI Hospitality</strong>!</p>
  </div>
</body>
</html>
"""

@app.route('/')
def home():
    return render_template_string(home_page)

@app.route('/book', methods=['POST'])
def book():
    name = request.form['name']
    room_type = request.form['room_type']
    checkin = request.form['checkin']
    checkout = request.form['checkout']
    return render_template_string(confirmation_page, name=name, room_type=room_type, checkin=checkin, checkout=checkout)

if __name__ == '__main__':
```



```
def home():
    return render_template_string(home_page)

@app.route('/book', methods=['POST'])
def book():
    name = request.form['name']
    room_type = request.form['room_type']
    checkin = request.form['checkin']
    checkout = request.form['checkout']
    return render_template_string(confirmation_page, name=name, room_type=room_type, checkin=checkin, checkout=checkout)

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

OUTPUT:



🏨 AI Hospitality Booking

HARINI

Room Type:

Single Room - ₹2000/night

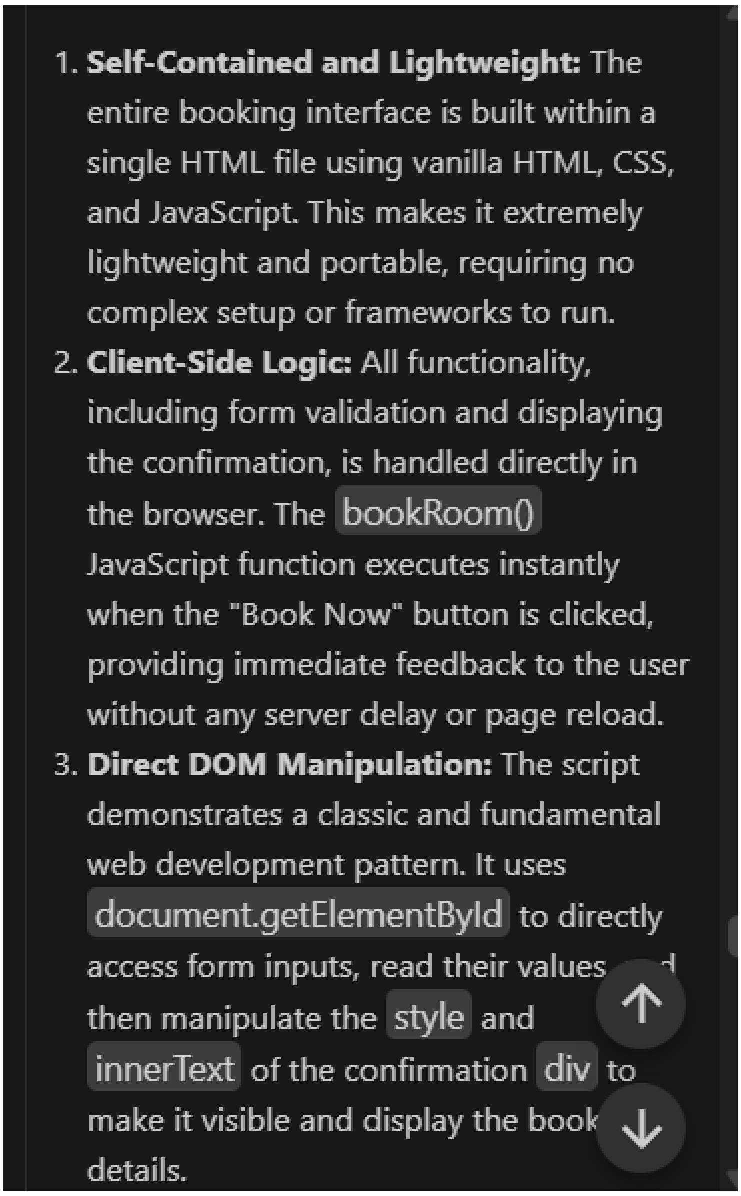
Check-in:

mm/dd/yyyy

Check-out:

mm/dd/yyyy

OBSERVATION:



1. **Self-Contained and Lightweight:** The entire booking interface is built within a single HTML file using vanilla HTML, CSS, and JavaScript. This makes it extremely lightweight and portable, requiring no complex setup or frameworks to run.
2. **Client-Side Logic:** All functionality, including form validation and displaying the confirmation, is handled directly in the browser. The `bookRoom()` JavaScript function executes instantly when the "Book Now" button is clicked, providing immediate feedback to the user without any server delay or page reload.
3. **Direct DOM Manipulation:** The script demonstrates a classic and fundamental web development pattern. It uses `document.getElementById` to directly access form inputs, read their values then manipulate the `style` and `innerText` of the confirmation `div` to make it visible and display the book details.