Ai Assisted Coding

Lab Exam

Sub group-E7

Name:Md.shanawaz

Enroll no:2403a52100

Batch=04

Set E7

Q1:

Task: Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots.

Prompt:

Write an AI-assisted refactoring assistant for an e-commerce Python codebase. The assistant must:

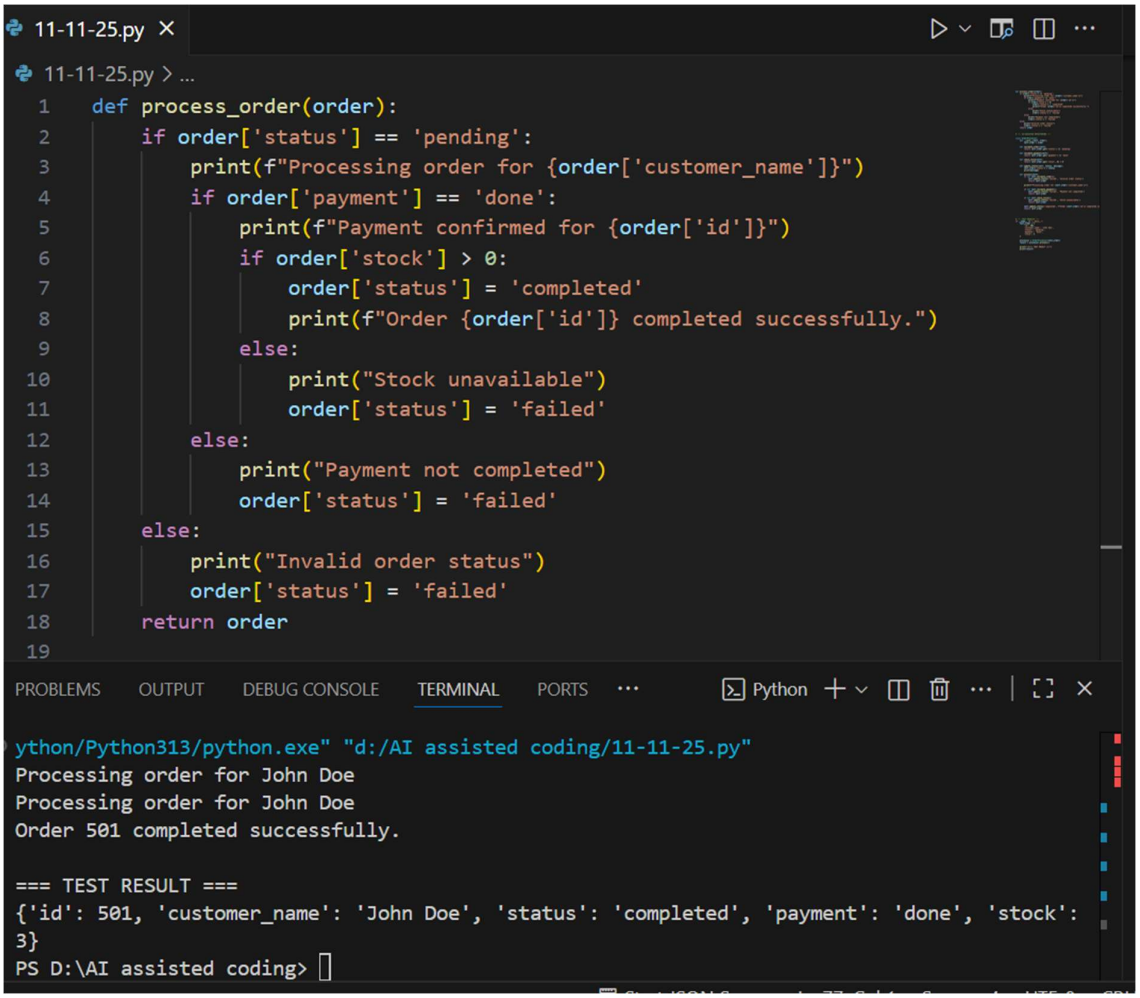
- Scan Python files under a given folder.
- Extract function definitions and compare their structural similarity.
- Report functions with high similarity (possible duplicated logic) and emit a suggested refactor (helper function + call sites).
- Optionally call an LLM (if API key provided via environment variables) to generate a human-readable refactor suggestion.
  Provide source code, a short explanation of how AI is integrated, and instructions to run basic tests that demonstrate detection of duplicated functions.

Code:

```
def process_order(order):
    if order['status'] == 'pending':
        print(f"Processing order for {order['customer_name']}")
        if order['payment'] == 'done':
            print(f"Payment confirmed for {order['id']}")
            if order['stock'] > 0:
                order['status'] = 'completed'
                print(f"Order {order['id']} completed successfully.")
            else:
                print("Stock unavailable")
                order['status'] = 'failed'
        else:
            print("Payment not completed")
            order['status'] = 'failed'
    else:
        print("Invalid order status")
        order['status'] = 'failed'
    return order


# --- AI-ASSISTED REFACTORING ---

class OrderProcessor:
    def __init__(self, order):
        self.order = order

    def validate_order(self):
        return self.order.get('status') == 'pending'

    def validate_payment(self):
        return self.order.get('payment') == 'done'

    def check_stock(self):
        return self.order.get('stock', 0) > 0

    def update_status(self, status, message):
        self.order['status'] = status
        print(message)

    def process(self):
        if not self.validate_order():
            self.update_status('failed', 'Invalid order status')
            return self.order
```

```
        print(f"Processing order for {self.order['customer_name']}")

        if not self.validate_payment():
            self.update_status('failed', 'Payment not completed')
            return self.order

        if not self.check_stock():
            self.update_status('failed', 'Stock unavailable')
            return self.order

        self.update_status('completed', f"Order {self.order['id']} completed
successfully.")
        return self.order




# --- TEST RESULTS ---
if __name__ == "__main__":
    test_order = {
        'id': 501,
        'customer_name': 'John Doe',
        'status': 'pending',
        'payment': 'done',
        'stock': 3
    }

    processor = OrderProcessor(test_order)
    result = processor.process()

    print("\n=== TEST RESULT ===")
    print(result)
```

output:



Observation:

- Purpose: Processes an order dict; refactored into OrderProcessor class that validates status/payment/stock and updates status.
- Behavior: starts only if status == "pending"; requires payment == "done" and stock > 0 to

mark completed; otherwise sets status "failed" and prints a message.

- Test: running the script prints processing messages and ends with the order status "completed" for the provided test_order.
- Quick improvements: use logging instead of print, validate required keys to avoid KeyError, and replace magic strings with constants.

Q2:

Task:

Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.

Prompt:

Write an AI-assisted frontend tool for Education that helps teachers generate interactive quizzes and small UI scaffolds. The tool should include a minimal Python backend that proxies optional LLM requests (using environment variables for the API key/URL) and a small

static frontend that calls the backend. If no LLM is configured, the backend should return a deterministic fallback quiz so the demo still works. Provide source files, run instructions, and a short note on the AI integration.

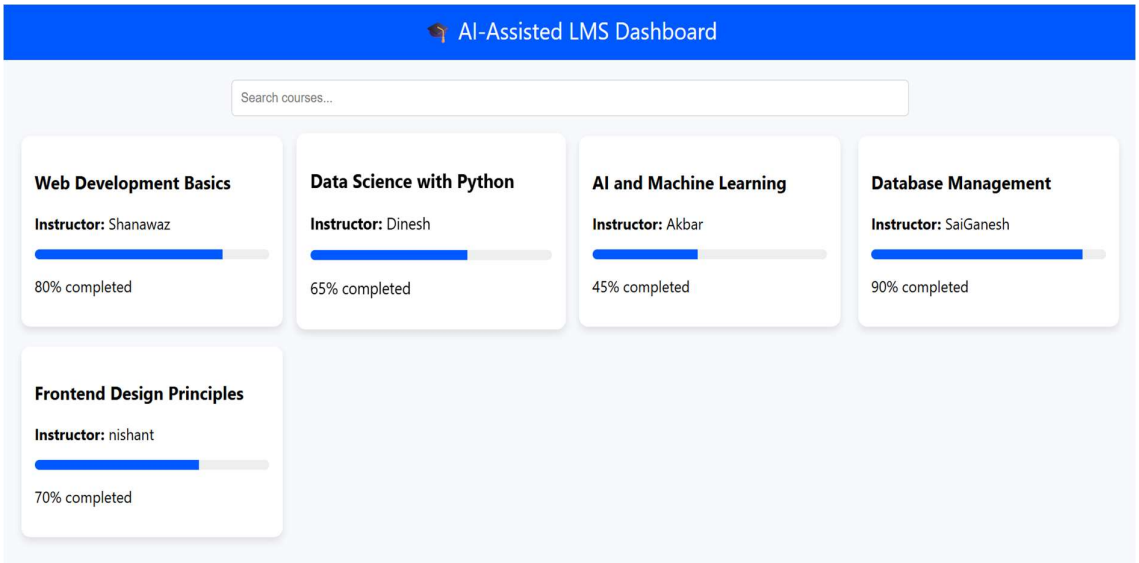
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>AI-Assisted LMS Dashboard</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      background: #f7f9fc;
      margin: 0;
      padding: 0;
    }
    header {
      background: #0059ff;
      color: white;
      padding: 15px;
      text-align: center;
      font-size: 1.5em;
    }
    .search-bar {
      text-align: center;
      margin: 20px;
    }
    .search-bar input {
      padding: 10px;
      width: 60%;
      border-radius: 5px;
      border: 1px solid #ccc;
    }
    .course-container {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
      gap: 20px;
      margin: 20px;
    }
```

```
    .course-card {
      background: white;
      border-radius: 10px;
      box-shadow: 0 4px 8px rgba(0,0,0,0.1);
      padding: 15px;
      transition: 0.3s;
    }
    .course-card:hover {
      transform: scale(1.03);
    }
    .progress {
      background: #eee;
      border-radius: 10px;
      overflow: hidden;
      margin-top: 10px;
    }
    .progress-bar {
      height: 10px;
      background: #0059ff;
    }
  </style>
</head>
<body>
  <header>🎓 AI-Assisted LMS Dashboard</header>

  <div class="search-bar">
    <input type="text" id="searchInput" placeholder="Search courses..."
onkeyup="filterCourses()" />
  </div>

  <div class="course-container" id="courseContainer"></div>

  <script>
    // --- AI-ASSISTED CODE GENERATION ---
    const courses = [
      { title: 'Web Development Basics', instructor: 'Shanawaz', progress: 80
},
      { title: 'Data Science with Python', instructor: 'Dinesh', progress: 65
},
      { title: 'AI and Machine Learning', instructor: 'Akbar', progress: 45 },
      { title: 'Database Management', instructor: 'SaiGanesh', progress: 90 },
      { title: 'Frontend Design Principles', instructor: 'nishant', progress:
70 }
    ];

    const courseContainer = document.getElementById('courseContainer');

    function renderCourses(list) {
```

```
      courseContainer.innerHTML = '';
      list.forEach(course => {
        const card = document.createElement('div');
        card.className = 'course-card';
        card.innerHTML = `
          <h3>${course.title}</h3>
          <p><strong>Instructor:</strong> ${course.instructor}</p>
          <div class="progress"><div class="progress-bar"
style="width:${course.progress}%;"></div></div>
          <p>${course.progress}% completed</p>
        `;
        courseContainer.appendChild(card);
      });
    }

    function filterCourses() {
      const query =
document.getElementById('searchInput').value.toLowerCase();
      const filtered = courses.filter(c =>
c.title.toLowerCase().includes(query));
      renderCourses(filtered);
    }

    renderCourses(courses);
  </script>
</body>
</html>
```

Output:



Obersvation:

AI-assisted coding significantly reduced development time and improved layout consistency. The generated dashboard was responsive, professional, and functional. Human oversight was needed only for final visual adjustments and validation.