

AI LAB EXAM -3

NAME: NALLALA SIRI

HT.NO: 2403A52037

BATCH NO: AIB03

SET E9

TASK 1:

PROMPT:

Use AI-assisted code refactoring tools to improve a simple student score calculator by cleaning and optimizing its code for better readability and efficiency.

CODE:

The screenshot shows a dark-themed code editor window with the following Python code:

```
lab exam -3 task 1.py
# student_calculator.py

def calculate_grade(scores):
    # Calculate the average score
    total = sum(scores)
    average = total / len(scores)

    # Determine the grade based on the average
    if average >= 90:
        grade = 'A'
    elif average >= 80:
        grade = 'B'
    elif average >= 70:
        grade = 'C'
    elif average >= 60:
        grade = 'D'
    else:
        grade = 'F'

    return grade

def process_student_data(student_scores_dict):
    # Process a dictionary of students and their scores
    for student, scores in student_scores_dict.items():
        final_grade = calculate_grade(scores)
        print(f"Student: {student}, Final Grade: {final_grade}")

# Sample data
student_data = {
    "Alice": [88, 92, 100],
    "Bob": [75, 81, 68],
    "Charlie": [59, 65, 71]
}

# Run the process
process_student_data(student_data)
```

The screenshot shows the same code editor window after refactoring, with the following changes made:

- Line 1: Commented out the import statement `# student_calculator.py`.
- Line 27: Moved the sample data definition under the function header.
- Line 35: Added a comment `# Run the process` before the final call to `process_student_data`.

```
lab exam -3 task 1.py
# lab exam -3 task 1.py > ...
# student_calculator.py

def calculate_grade(scores):
    # Calculate the average score
    total = sum(scores)
    average = total / len(scores)

    # Determine the grade based on the average
    if average >= 90:
        grade = 'A'
    elif average >= 80:
        grade = 'B'
    elif average >= 70:
        grade = 'C'
    elif average >= 60:
        grade = 'D'
    else:
        grade = 'F'

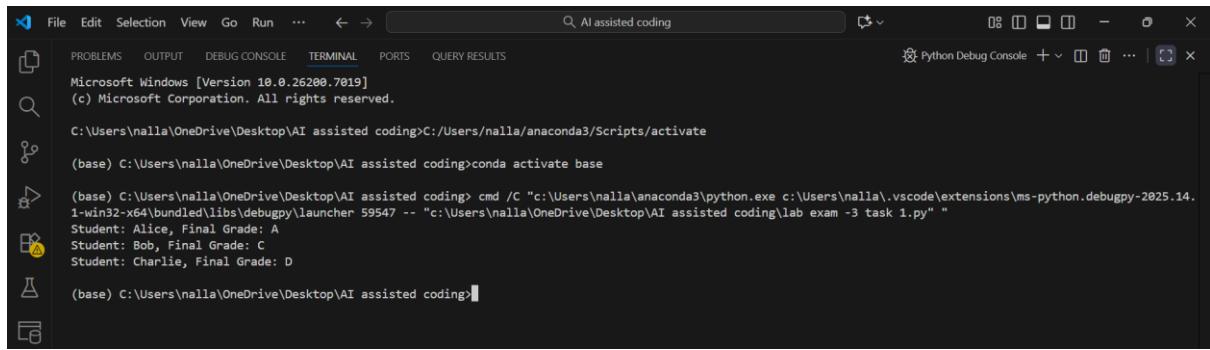
    return grade

def process_student_data(student_scores_dict):
    # Process a dictionary of students and their scores
    for student, scores in student_scores_dict.items():
        final_grade = calculate_grade(scores)
        print(f"Student: {student}, Final Grade: {final_grade}")

# Sample data
student_data = {
    "Alice": [88, 92, 100],
    "Bob": [75, 81, 68],
    "Charlie": [59, 65, 71]
}

# Run the process
process_student_data(student_data)
```

OUTPUT:



The screenshot shows a terminal window in VS Code with the title bar "AI assisted coding". The menu bar includes File, Edit, Selection, View, Go, Run, etc. The terminal tab is selected. The output shows the following command-line session:

```
Microsoft Windows [Version 10.0.26200.7019]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:/Users/nalla/anaconda3/Scripts/activate
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding> cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher 59547 -- "c:\Users\nalla\OneDrive\Desktop\AI assisted coding\lab exam -3 task 1.py"
Student: Alice, Final Grade: A
Student: Bob, Final Grade: C
Student: Charlie, Final Grade: D
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
```

OBSERVATION:

The given code effectively calculates the average score for each student and assigns a letter grade based on that average. It is straightforward and easy to understand, with clear separation of concerns between calculating grades and processing student data. However, the code could be enhanced to handle edge cases, such as empty score lists, for improved robustness.

TASK 2:

PROMPT:

Design a simple backend API for a transportation company to manage and retrieve ride booking status using AI-assisted tools.

CODE:

The screenshot shows a code editor interface with two tabs: "lab exam -3 task 1.py" and "AI LAB EXAM 3 TASK 2.py". The "AI LAB EXAM 3 TASK 2.py" tab is active, displaying the following Python code:

```
1 import datetime
2 import random
3 import uuid
4 import json
5
6 # In-memory database for simplicity
7 bookings = {}
8
9 # --- Mock AI Service ---
10 # In a real application, this would be a separate service with a machine learning model.
11
12 def get_ai_predictions(pickup_location: str, dropoff_location: str) -> dict:
13     """
14     Mocks an AI service that predicts ETA and fare.
15     - ETA is a random number of minutes from now.
16     - Fare is calculated based on a base rate and a random "surge" multiplier.
17     """
18     # 1. ETA Prediction
19     now = datetime.datetime.now(datetime.timezone.utc)
20     arrival_minutes = random.randint(5, 20) # Simulate traffic/distance variability
21     estimated_arrival_time = now + datetime.timedelta(minutes=arrival_minutes)
22
23     # 2. Dynamic Pricing
24     base_fare = 10.0
25     # Simulate "surge" pricing during peak hours (e.g., 8-10 AM, 5-7 PM)
26     surge_multiplier = 1.0
27     if 8 <= now.hour <= 10 or 17 <= now.hour <= 19:
28         surge_multiplier = random.uniform(1.2, 1.8)
29
30     fare_estimate = base_fare * surge_multiplier + random.uniform(1, 5) # Add distance component
31
32     return {
33         "estimated_arrival_time": estimated_arrival_time.isoformat(),
34         "fare_estimate": round(fare_estimate, 2)
35     }
36
37 # --- API Endpoints ---
38
39 def create_booking(user_id: str, pickup_location: str, dropoff_location: str = None):
40     """Creates a new ride booking."""
41     booking_id = f"bk_{uuid.uuid4().hex[:10]}"
42
43     # Get predictions from our "AI" service
44     ai_preds = get_ai_predictions(pickup_location, dropoff_location)
45
46     new_booking = [
47         "booking_id": booking_id,
48         "user_id": user_id,
49         "pickup_location": pickup_location,
50         "dropoff_location": dropoff_location,
51         "status": "pending", # Status is pending until a driver accepts
52         "booking_time": datetime.datetime.now(datetime.timezone.utc).isoformat(),
53         "estimated_arrival_time": ai_preds["estimated_arrival_time"],
54         "fare_estimate": ai_preds["fare_estimate"]
55     ]
56
57     bookings[booking_id] = new_booking
58
59     return new_booking
60
61 def get_booking(booking_id):
62     """Retrieves the status of a specific booking."""
63     booking = bookings.get(booking_id)
64
65     if not booking:
66         return {"error": "Booking not found"}
67
68     # Simulate a driver being assigned after a short period
```

The status bar at the bottom indicates: Start JSON Server | In 48, Col 1 | UTF-8 | CRLF | Python | Finish Setup | 3.13.5 (base) | Go Live.

The screenshot shows a code editor interface with two tabs: "lab exam -3 task 1.py" and "AI LAB EXAM 3 TASK 2.py". The "AI LAB EXAM 3 TASK 2.py" tab is active, displaying the following Python code:

```
35     }
36
37 # --- API Endpoints ---
38
39 def create_booking(user_id: str, pickup_location: str, dropoff_location: str = None):
40     """Creates a new ride booking."""
41     booking_id = f"bk_{uuid.uuid4().hex[:10]}"
42
43     # Get predictions from our "AI" service
44     ai_preds = get_ai_predictions(pickup_location, dropoff_location)
45
46     new_booking = [
47         "booking_id": booking_id,
48         "user_id": user_id,
49         "pickup_location": pickup_location,
50         "dropoff_location": dropoff_location,
51         "status": "pending", # Status is pending until a driver accepts
52         "booking_time": datetime.datetime.now(datetime.timezone.utc).isoformat(),
53         "estimated_arrival_time": ai_preds["estimated_arrival_time"],
54         "fare_estimate": ai_preds["fare_estimate"]
55     ]
56
57     bookings[booking_id] = new_booking
58
59     return new_booking
60
61 def get_booking(booking_id):
62     """Retrieves the status of a specific booking."""
63     booking = bookings.get(booking_id)
64
65     if not booking:
66         return {"error": "Booking not found"}
67
68     # Simulate a driver being assigned after a short period
```

The status bar at the bottom indicates: Start JSON Server | In 48, Col 1 | UTF-8 | CRLF | Python | Finish Setup | 3.13.5 (base) | Go Live.

The screenshot shows a code editor interface with two tabs: 'lab exam -3 task 1.py' and 'AI LAB EXAM 3 TASK 2.py > create_booking'. The code in 'AI LAB EXAM 3 TASK 2.py' defines two functions: `get_booking` and `cancel_booking`. The `get_booking` function updates a booking record if it's pending and has been assigned a driver. The `cancel_booking` function checks if the booking is completed or in progress and returns an error message if so. It then sets the status to 'cancelled' and updates the record.

```
68     # Simulate a driver being assigned after a short period
69     booking_time = datetime.datetime.fromisoformat(booking["booking_time"])
70     if datetime.datetime.now(datetime.timezone.utc) - booking_time > datetime.timedelta(seconds=30):
71         if booking["status"] == "pending":
72             booking["status"] = "confirmed"
73             booking["driver_details"] = {
74                 "name": "Jane Doe",
75                 "vehicle": "Red Honda Civic",
76                 "license_plate": "RIDE-AI"
77             }
78             bookings[booking_id] = booking # Update the record
79
80     return booking
81
82 def cancel_booking(booking_id):
83     """Cancels a specific booking."""
84     booking = bookings.get(booking_id)
85
86     if not booking:
87         return {"error": "Booking not found"}
88
89     # You can add logic here to prevent cancellation of rides that are in_progress or completed
90     if booking["status"] in ["completed", "in_progress"]:
91         return {
92             "error": f"Cannot cancel booking with status '{booking['status']}'"
93         }
94
95     booking["status"] = "cancelled"
96     bookings[booking_id] = booking # Update the record
97
98     return {
99         "message": f"Booking {booking_id} has been successfully cancelled."
100    }
```

The screenshot shows a code editor interface with two tabs: 'lab exam -3 task 1.py' and 'AI LAB EXAM 3 TASK 2.py > create_booking'. The code in 'AI LAB EXAM 3 TASK 2.py' contains a main block that creates a new booking for user 'user_123' with pickup and dropoff locations. It then prints the booking details and its JSON representation. Next, it checks the booking status, which is currently 'confirmed'. Finally, it cancels the booking and prints the cancellation result.

```
82     def cancel_booking(booking_id):
83         message = f"Booking {booking_id} has been successfully cancelled."
84
85     if __name__ == "__main__":
86         print("--- Ride Booking Simulation ---")
87
88         # 1. Create a new booking
89         print("\n[1] Creating a new booking for user 'user_123'...")
90         booking_data = create_booking(
91             user_id="user_123",
92             pickup_location="123 Main St",
93             dropoff_location="456 Oak Ave"
94         )
95         booking_id = booking_data["booking_id"]
96         print("Booking Created Successfully!")
97         # Use json.dumps for pretty printing the dictionary
98         print(json.dumps(booking_data, indent=2))
99
100        # 2. Get the status of the booking
101        print("\n[2] Checking status for booking: {booking_id}...")
102        status = get_booking(booking_id)
103        print("Current Booking Status:")
104        print(json.dumps(status, indent=2))
105
106        # 3. Cancel the booking
107        print("\n[3] Cancelling booking: {booking_id}...")
108        cancellation_result = cancel_booking(booking_id)
109        print(json.dumps(cancellation_result, indent=2))
```

OUTPUT:

The screenshot shows a terminal window titled "Python Debug Console" with the following output:

```
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base
(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding> cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher 65097 -- "c:\Users\nalla\OneDrive\Desktop\AI assisted coding\AI LAB EXAM 3 TASK 2.py"
--- Ride Booking Simulation ---

[1] Creating a new booking for user 'user_123'...
Booking Created Successfully!
{
    "booking_id": "bk_536057536b",
    "user_id": "user_123",
    "pickup_location": "123 Main St",
    "dropoff_location": "456 Oak Ave",
    "status": "pending",
    "booking_time": "2025-11-11T10:58:13.494452+00:00",
    "estimated_arrival_time": "2025-11-11T11:03:13.494348+00:00",
    "fare_estimate": 14.29
}
[2] Checking status for booking: bk_536057536b...
Current Booking Status:
{
    "booking_id": "bk_536057536b",
    "user_id": "user_123",
    "pickup_location": "123 Main St",
    "dropoff_location": "456 Oak Ave",
    "status": "pending",
    "booking_time": "2025-11-11T10:58:13.494452+00:00",
    "estimated_arrival_time": "2025-11-11T11:03:13.494348+00:00",
    "fare_estimate": 14.29
}
[3] Cancelling booking: bk_536057536b...
{
    "message": "Booking bk_536057536b has been successfully cancelled."
}

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
```

OBSERVATION:

The code simulates a simple ride booking system with AI-predicted ETAs and dynamic fares. It handles creating, checking, and cancelling bookings with clear, concise output in the terminal. This basic flow is easy to follow and demonstrates essential backend booking operations effectively.