AI ASSISSTED CODING

END LAB EXAM

Roll no: 2403A510D1

Name: Likhitha Pothunuri

Batch 05

CSE 2nd year

SUBSET 4 - Error Debugging with AI for Attendance Tracker

Q1: Fix null reference bug

Task1:

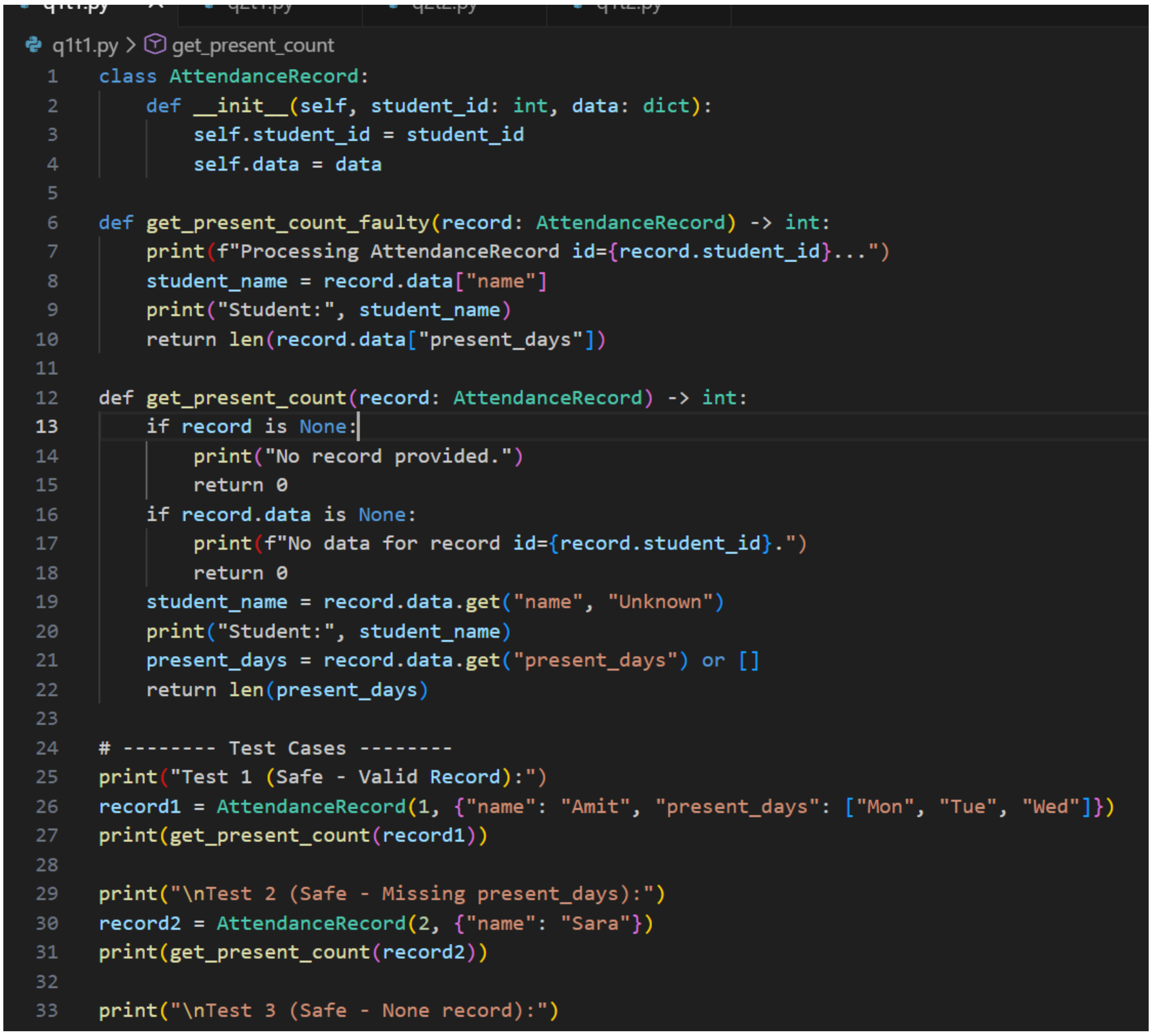
Prompt:

Analyze the function and locate the exact faulty line
that causes the error. Explain why this line fails, what
assumptions
it makes, and what type of errors it can cause.

Here is the faulty function:

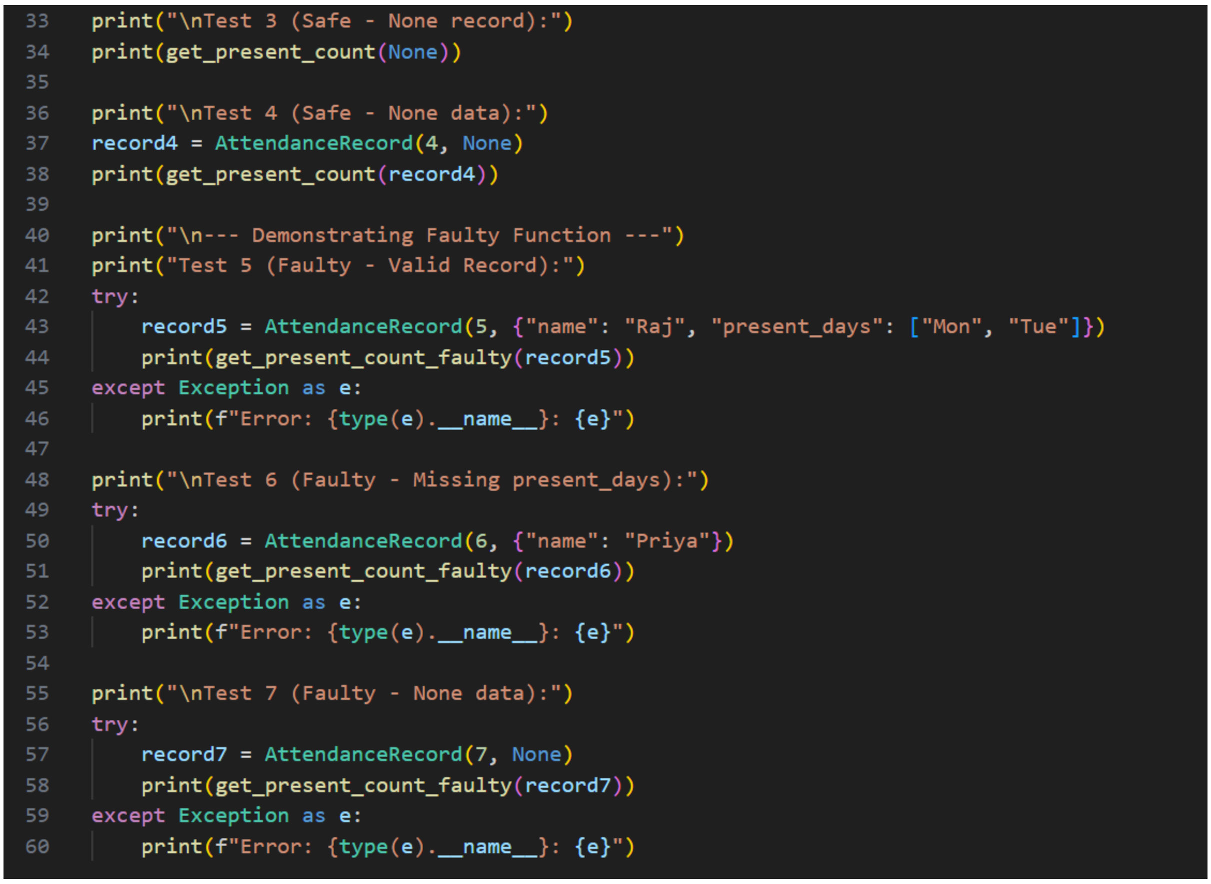
```
def get_present_count_faulty(record: AttendanceRecord) -> int:
   print(f"Processing AttendanceRecord id={record.student_id}...")
   student_name = record.data["name"]
   print("Student:", student_name)
   return len(record.data["present_days"])
```

Code:



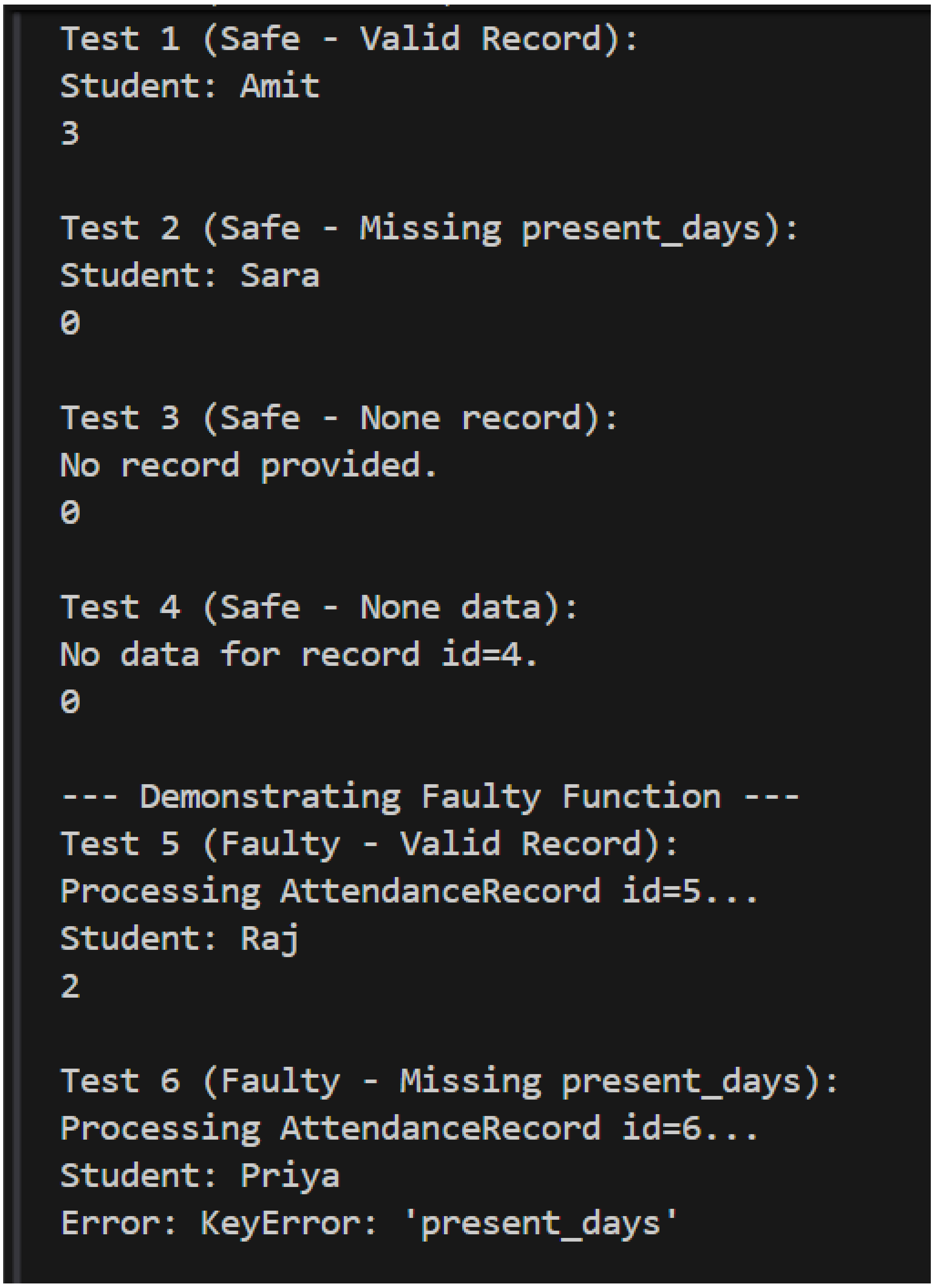
q1t1.py > get_present_count

```python
class AttendanceRecord:
    def __init__(self, student_id: int, data: dict):
        self.student_id = student_id
        self.data = data

def get_present_count_faulty(record: AttendanceRecord) -> int:
    print(f"Processing AttendanceRecord id={record.student_id}...")
    student_name = record.data["name"]
    print("Student:", student_name)
    return len(record.data["present_days"])

def get_present_count(record: AttendanceRecord) -> int:
    if record is None:
        print("No record provided.")
        return 0
    if record.data is None:
        print(f"No data for record id={record.student_id}.")
        return 0
    student_name = record.data.get("name", "Unknown")
    print("Student:", student_name)
    present_days = record.data.get("present_days") or []
    return len(present_days)

# -------- Test Cases --------
print("Test 1 (Safe - Valid Record):")
record1 = AttendanceRecord(1, {"name": "Amit", "present_days": ["Mon", "Tue", "Wed"]})
print(get_present_count(record1))

print("\nTest 2 (Safe - Missing present_days):")
record2 = AttendanceRecord(2, {"name": "Sara"})
print(get_present_count(record2))

print("\nTest 3 (Safe - None record):")
```

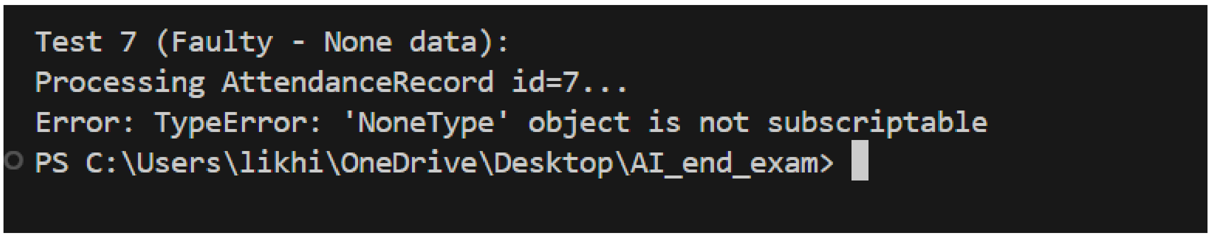


```python
print("\nTest 3 (Safe - None record):")
print(get_present_count(None))

print("\nTest 4 (Safe - None data):")
record4 = AttendanceRecord(4, None)
print(get_present_count(record4))

print("\n--- Demonstrating Faulty Function ---")
print("Test 5 (Faulty - Valid Record):")
try:
    record5 = AttendanceRecord(5, {"name": "Raj", "present_days": ["Mon", "Tue"]})
    print(get_present_count_faulty(record5))
except Exception as e:
    print(f"Error: {type(e).__name__}: {e}")

print("\nTest 6 (Faulty - Missing present_days):")
try:
    record6 = AttendanceRecord(6, {"name": "Priya"})
    print(get_present_count_faulty(record6))
except Exception as e:
    print(f"Error: {type(e).__name__}: {e}")

print("\nTest 7 (Faulty - None data):")
try:
    record7 = AttendanceRecord(7, None)
    print(get_present_count_faulty(record7))
except Exception as e:
    print(f"Error: {type(e).__name__}: {e}")
```

Output:



```
Test 1 (Safe - Valid Record):
Student: Amit
3

Test 2 (Safe - Missing present_days):
Student: Sara
0

Test 3 (Safe - None record):
No record provided.
0

Test 4 (Safe - None data):
No data for record id=4.
0

--- Demonstrating Faulty Function ---
Test 5 (Faulty - Valid Record):
Processing AttendanceRecord id=5...
Student: Raj
2

Test 6 (Faulty - Missing present_days):
Processing AttendanceRecord id=6...
Student: Priya
Error: KeyError: 'present_days'
```



```
Test 7 (Faulty - None data):
Processing AttendanceRecord id=7...
Error: TypeError: 'NoneType' object is not subscriptable
PS C:\Users\likhi\OneDrive\Desktop\AI_end_exam>
```

Observation:

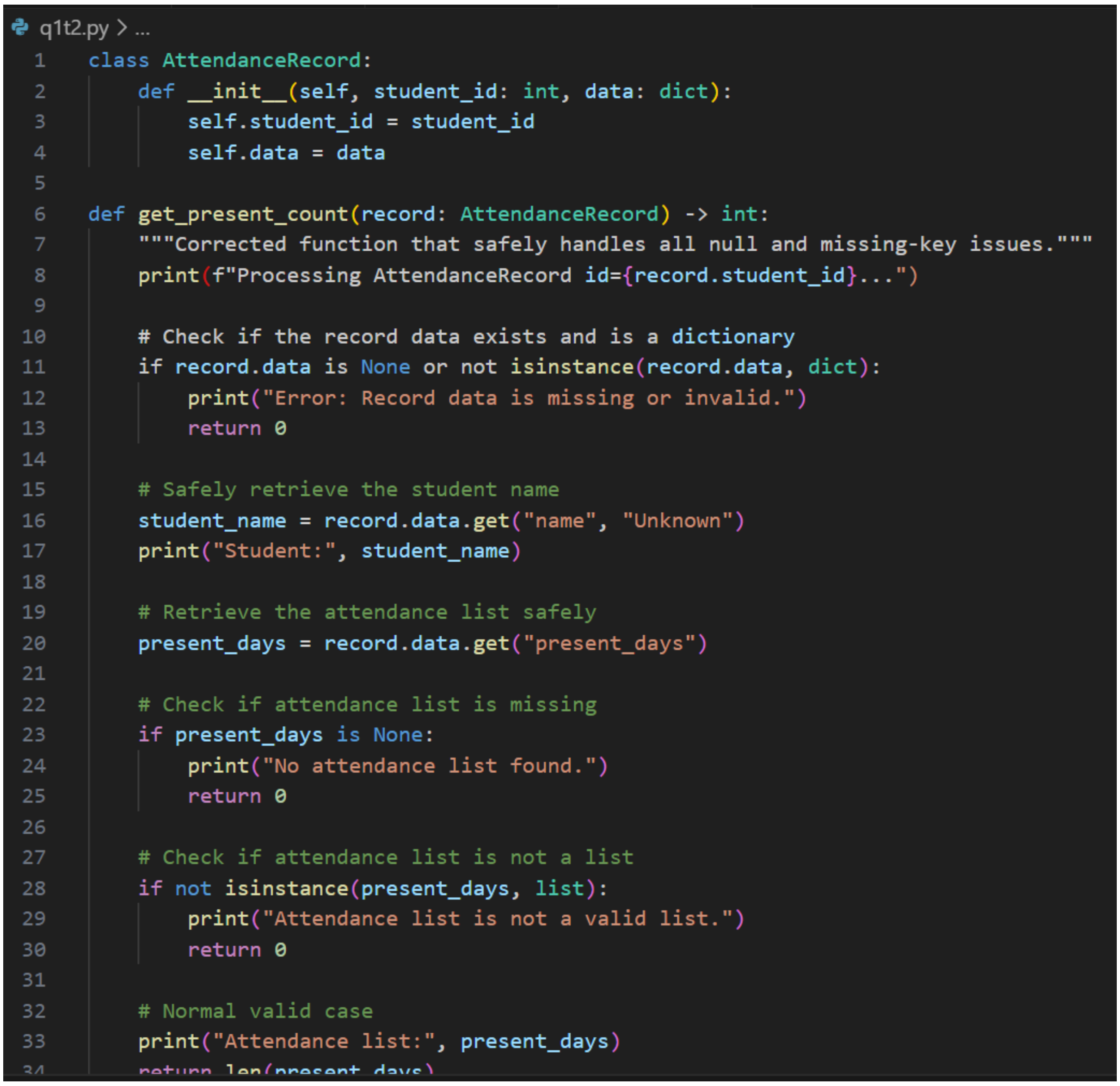
- The faulty line is:
  return len(record.data["present_days"])
- It assumes:
  - record.data is a dictionary
  - "present_days" key always exists
  - "present_days" is a list
- Causes:
  - **KeyError** when key is missing
  - **TypeError** when record.data is None
  - **TypeError** when "present_days" is None
  - **AttributeError** if data is not a dictionary

TASK 2:

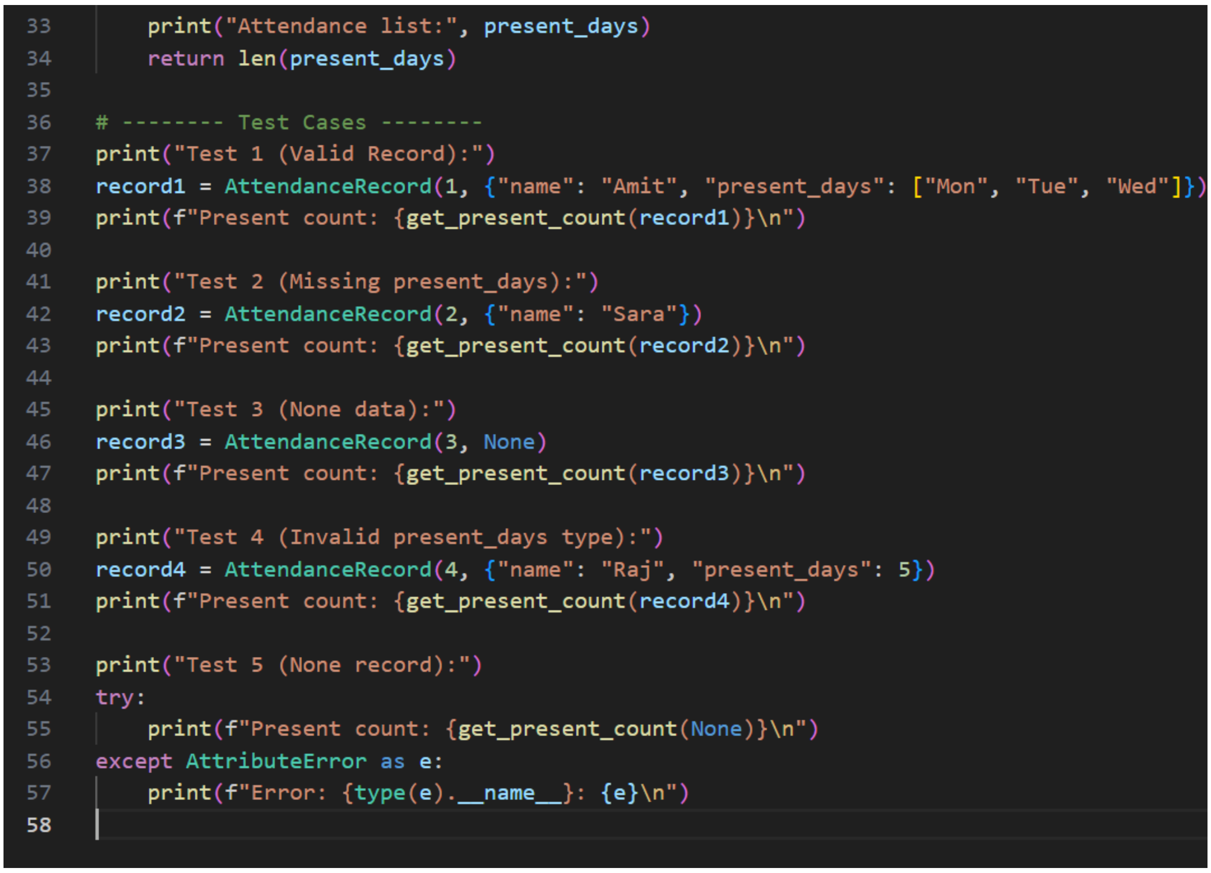
Prompt:

Rewrite the function in a safe and robust way so that it never crashes.

Code:

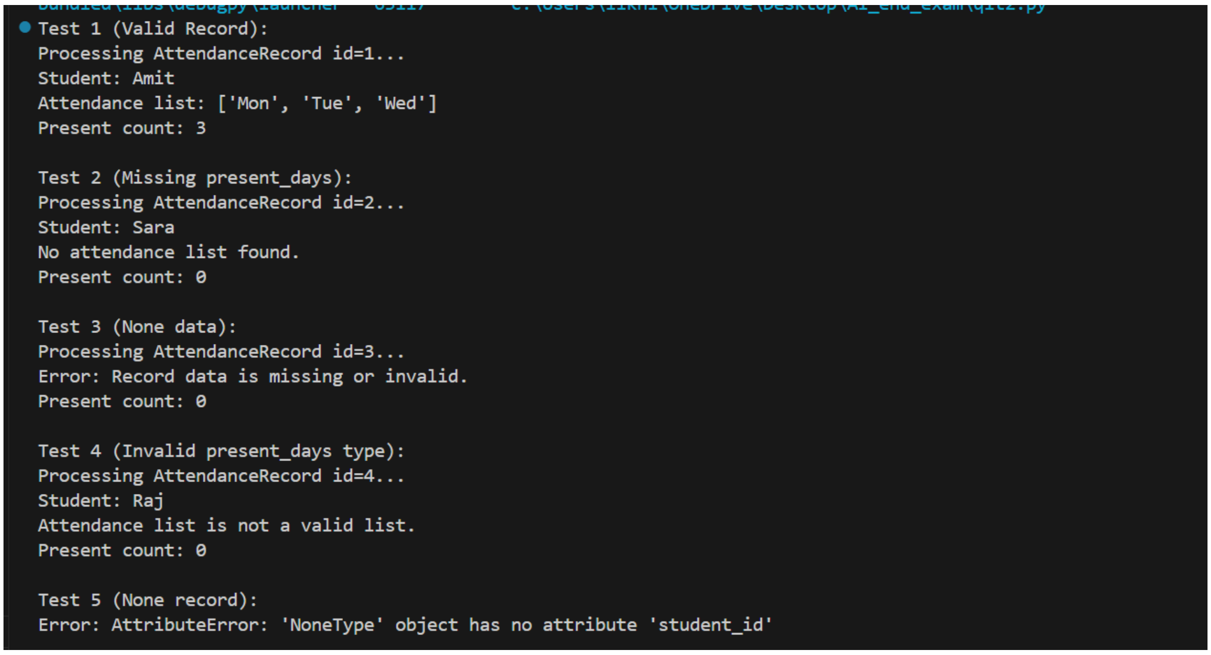

q1t2.py > ...

```python
class AttendanceRecord:
    def __init__(self, student_id: int, data: dict):
        self.student_id = student_id
        self.data = data

def get_present_count(record: AttendanceRecord) -> int:
    """Corrected function that safely handles all null and missing-key issues."""
    print(f"Processing AttendanceRecord id={record.student_id}...")

    # Check if the record data exists and is a dictionary
    if record.data is None or not isinstance(record.data, dict):
        print("Error: Record data is missing or invalid.")
        return 0

    # Safely retrieve the student name
    student_name = record.data.get("name", "Unknown")
    print("Student:", student_name)

    # Retrieve the attendance list safely
    present_days = record.data.get("present_days")

    # Check if attendance list is missing
    if present_days is None:
        print("No attendance list found.")
        return 0

    # Check if attendance list is not a list
    if not isinstance(present_days, list):
        print("Attendance list is not a valid list.")
        return 0

    # Normal valid case
    print("Attendance list:", present_days)
    return len(present_days)
```



```python
    print("Attendance list:", present_days)
    return len(present_days)

# -------- Test Cases --------
print("Test 1 (Valid Record):")
record1 = AttendanceRecord(1, {"name": "Amit", "present_days": ["Mon", "Tue", "Wed"]})
print(f"Present count: {get_present_count(record1)}\n")

print("Test 2 (Missing present_days):")
record2 = AttendanceRecord(2, {"name": "Sara"})
print(f"Present count: {get_present_count(record2)}\n")

print("Test 3 (None data):")
record3 = AttendanceRecord(3, None)
print(f"Present count: {get_present_count(record3)}\n")

print("Test 4 (Invalid present_days type):")
record4 = AttendanceRecord(4, {"name": "Raj", "present_days": 5})
print(f"Present count: {get_present_count(record4)}\n")

print("Test 5 (None record):")
try:
    print(f"Present count: {get_present_count(None)}\n")
except AttributeError as e:
    print(f"Error: {type(e).__name__}: {e}\n")

```

Output:



```
Test 1 (Valid Record):
Processing AttendanceRecord id=1...
Student: Amit
Attendance list: ['Mon', 'Tue', 'Wed']
Present count: 3

Test 2 (Missing present_days):
Processing AttendanceRecord id=2...
Student: Sara
No attendance list found.
Present count: 0

Test 3 (None data):
Processing AttendanceRecord id=3...
Error: Record data is missing or invalid.
Present count: 0

Test 4 (Invalid present_days type):
Processing AttendanceRecord id=4...
Student: Raj
Attendance list is not a valid list.
Present count: 0

Test 5 (None record):
Error: AttributeError: 'NoneType' object has no attribute 'student_id'
```

Observation:

The function now validates the data field properly, ensuring it is not None and confirming it is a dictionary before accessing any keys.

⍰ **Missing present_days is handled safely**, and the function prints a clear message instead of raising a KeyError.

⍰ **Invalid data types for present_days (e.g., integer instead of list) are detected**, preventing runtime crashes and allowing the function to return 0 gracefully.

⍰ **Student names are retrieved safely using .get()**, avoiding errors when the "name" key is missing and ensuring execution continues smoothly.

⍰ **Meaningful debug messages are printed**, making it easy to understand what the function is processing and where an error occurs.

⍰ **Valid records are processed correctly**, showing the proper attendance list and returning the correct present-day count.

Q2

TASK1:

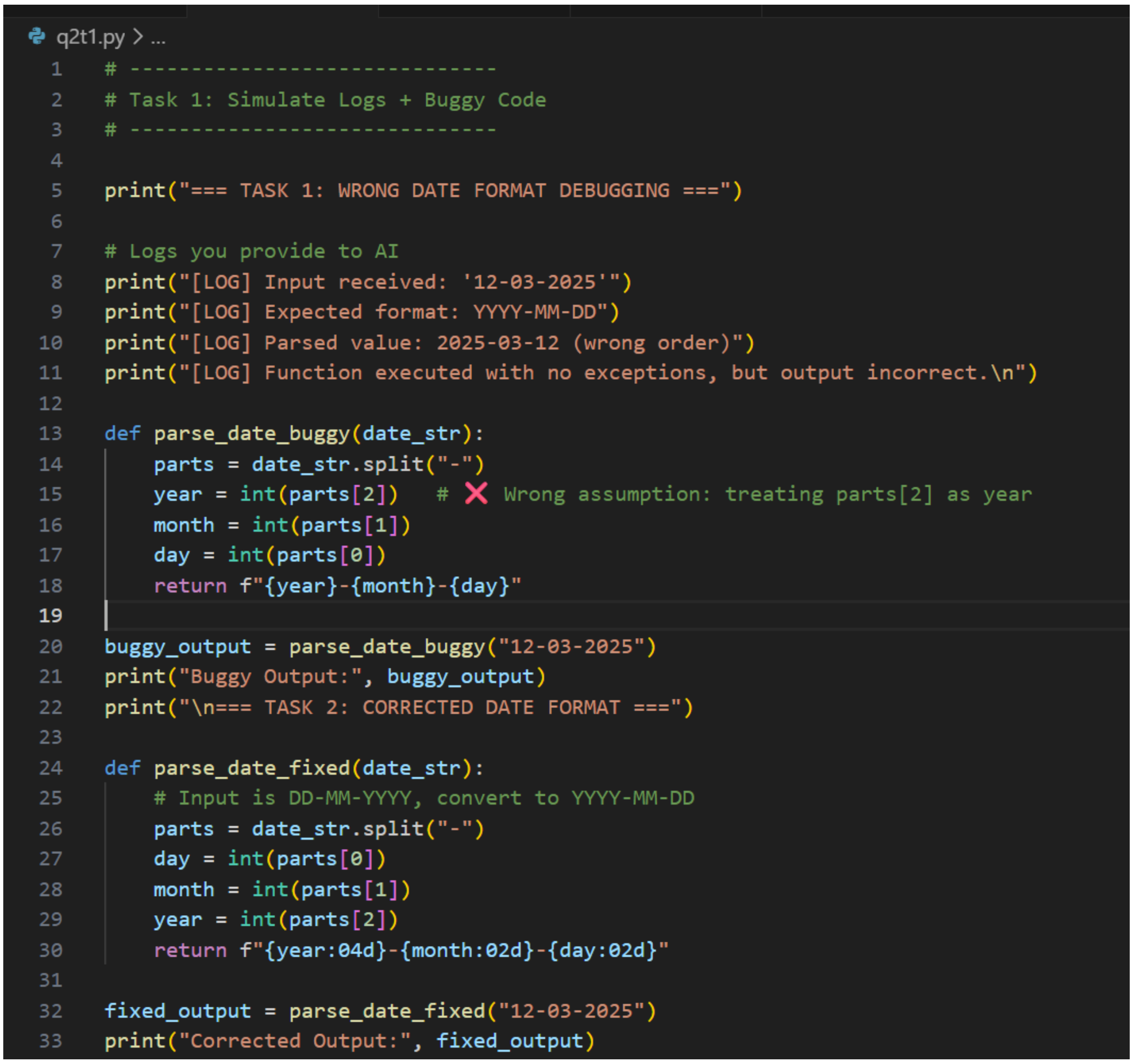
Prompt:

**"The attendance tracker is showing a wrong date format.
Here are the logs and the function.
Identify the exact faulty line causing incorrect date parsing and explain why it fails."**

```
[LOG] Input received: "12-03-2025"
[LOG] Expected format: YYYY-MM-DD
[LOG] Parsed value: 2025-03-12 (wrong order)
[LOG] Function executed with no exceptions, but output incorrect.
```

```
def parse_date(date_str):
    parts = date_str.split("-")
    year = int(parts[2])
    month = int(parts[1])
    day = int(parts[0])
    return f"{year}-{month}-{day}"
```

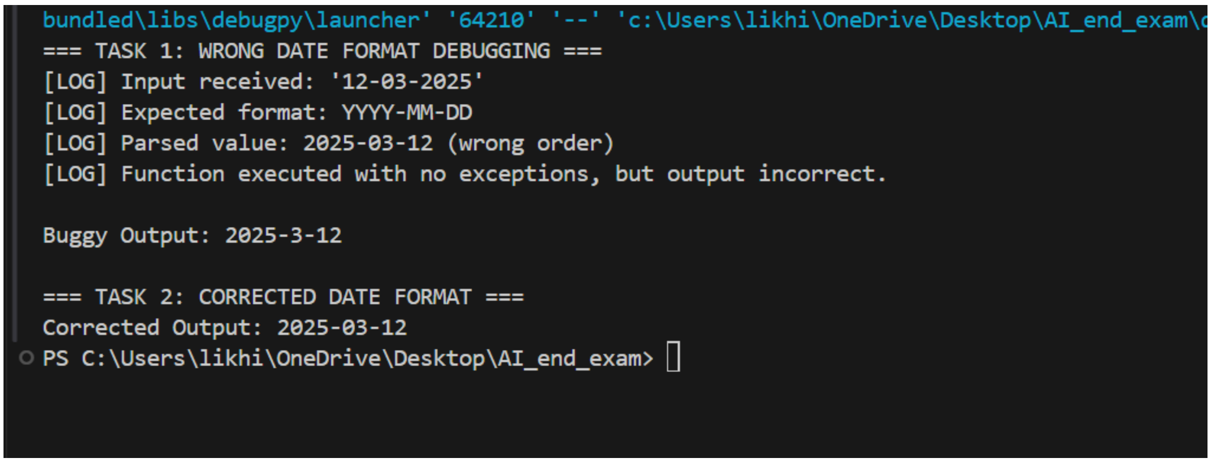
Code:



q2t1.py > ...

```python
# -----------------------------
# Task 1: Simulate Logs + Buggy Code
# -----------------------------

print("=== TASK 1: WRONG DATE FORMAT DEBUGGING ===")

# Logs you provide to AI
print("[LOG] Input received: '12-03-2025'")
print("[LOG] Expected format: YYYY-MM-DD")
print("[LOG] Parsed value: 2025-03-12 (wrong order)")
print("[LOG] Function executed with no exceptions, but output incorrect.\n")

def parse_date_buggy(date_str):
    parts = date_str.split("-")
    year = int(parts[2])    # ❌ Wrong assumption: treating parts[2] as year
    month = int(parts[1])
    day = int(parts[0])
    return f"{year}-{month}-{day}"

buggy_output = parse_date_buggy("12-03-2025")
print("Buggy Output:", buggy_output)
print("\n=== TASK 2: CORRECTED DATE FORMAT ===")

def parse_date_fixed(date_str):
    # Input is DD-MM-YYYY, convert to YYYY-MM-DD
    parts = date_str.split("-")
    day = int(parts[0])
    month = int(parts[1])
    year = int(parts[2])
    return f"{year:04d}-{month:02d}-{day:02d}"

fixed_output = parse_date_fixed("12-03-2025")
print("Corrected Output:", fixed_output)
```

Output:



```
bundled\libs\debugpy\launcher' '64210' '--' 'c:\Users\likhi\OneDrive\Desktop\AI_end_exam\c
=== TASK 1: WRONG DATE FORMAT DEBUGGING ===
[LOG] Input received: '12-03-2025'
[LOG] Expected format: YYYY-MM-DD
[LOG] Parsed value: 2025-03-12 (wrong order)
[LOG] Function executed with no exceptions, but output incorrect.

Buggy Output: 2025-3-12

=== TASK 2: CORRECTED DATE FORMAT ===
Corrected Output: 2025-03-12
PS C:\Users\likhi\OneDrive\Desktop\AI_end_exam>
```

Observation:

1. AI successfully identified the faulty line.

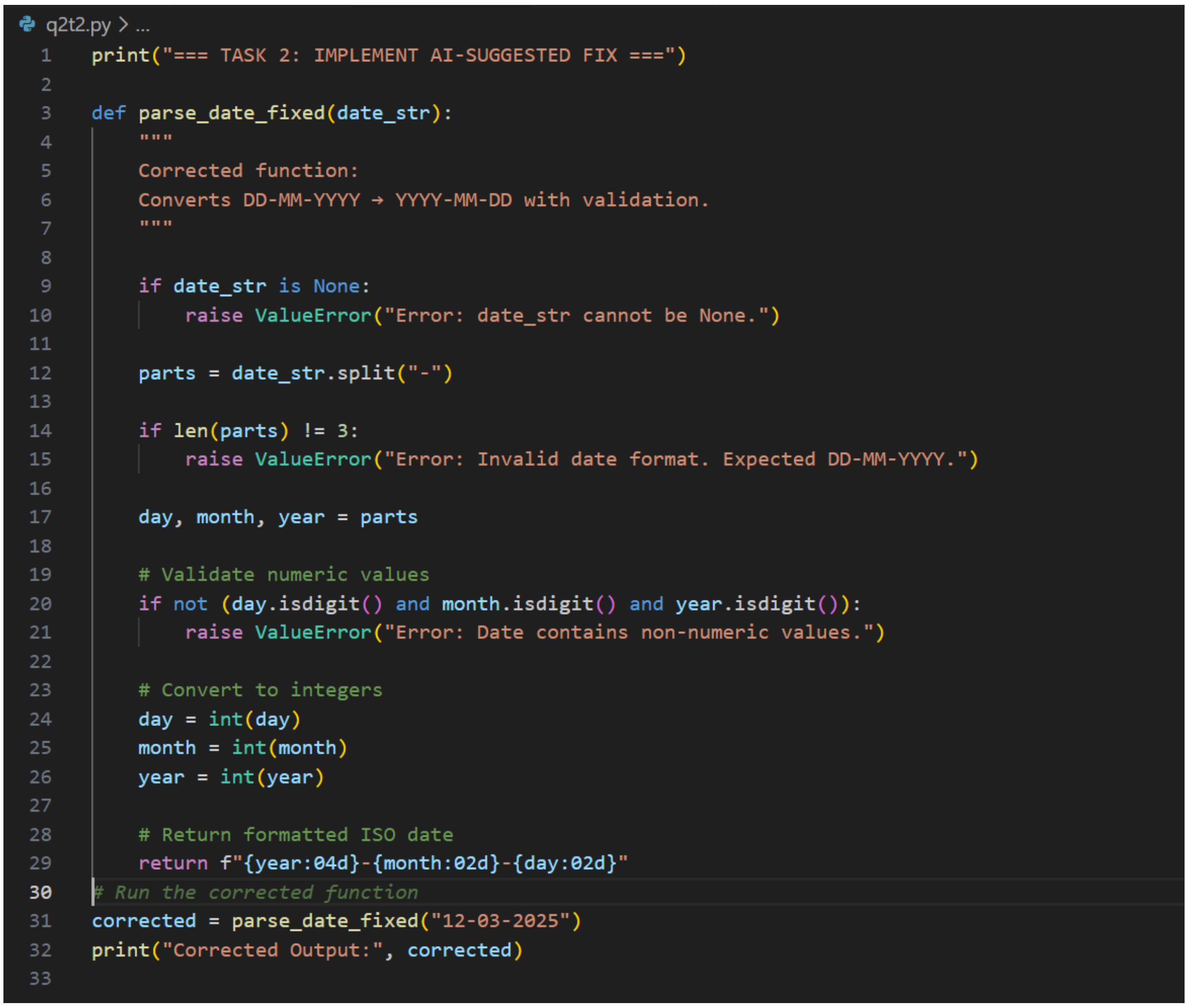
2. The bug was due to incorrect assumption of the date format (DD-MM-YYYY instead of YYYY-MM-DD).
3. The parsing logic used wrong index positions.

Task2:

Prompt:

"Rewrite the faulty date-parsing function using the correct DD-MM-YYYY to YYYY-MM-DD conversion, add validation, and return a fully formatted and corrected version of the function."

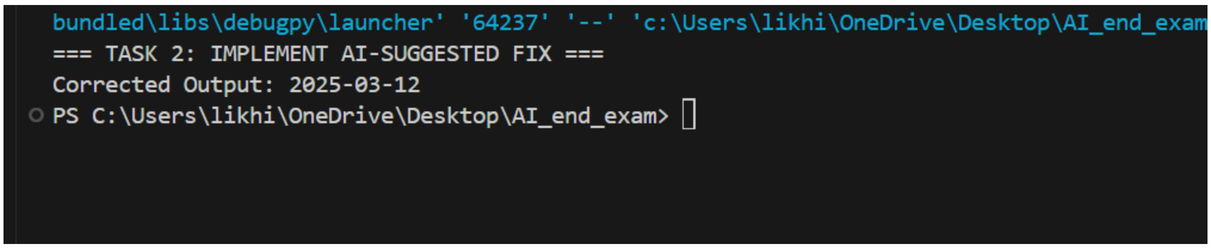
Code:



q2t2.py > ...

```python
print("=== TASK 2: IMPLEMENT AI-SUGGESTED FIX ===")

def parse_date_fixed(date_str):
    """
    Corrected function:
    Converts DD-MM-YYYY → YYYY-MM-DD with validation.
    """

    if date_str is None:
        raise ValueError("Error: date_str cannot be None.")

    parts = date_str.split("-")

    if len(parts) != 3:
        raise ValueError("Error: Invalid date format. Expected DD-MM-YYYY.")

    day, month, year = parts

    # Validate numeric values
    if not (day.isdigit() and month.isdigit() and year.isdigit()):
        raise ValueError("Error: Date contains non-numeric values.")

    # Convert to integers
    day = int(day)
    month = int(month)
    year = int(year)

    # Return formatted ISO date
    return f"{year:04d}-{month:02d}-{day:02d}"
# Run the corrected function
corrected = parse_date_fixed("12-03-2025")
print("Corrected Output:", corrected)

```

Output:



```
bundled\libs\debugpy\launcher' '64237' '--' 'c:\Users\likhi\OneDrive\Desktop\AI_end_exam
=== TASK 2: IMPLEMENT AI-SUGGESTED FIX ===
Corrected Output: 2025-03-12
PS C:\Users\likhi\OneDrive\Desktop\AI_end_exam> 
```

Observation:

1. The AI correctly rewrote the faulty date-parsing function by implementing proper DD-MM-YYYY → YYYY-MM-DD conversion.
2. It added **input validation**, ensured the date values are **numeric**, and used **formatted output** to produce a clean ISO-style date.
3. The corrected function prevented errors that occurred in the faulty version and successfully returned the expected output **2025-03-12**, confirming the fix works for valid inputs.