AI - Assisted Coding-Assignment -7

HTNO:2303A52228
Batch:37

Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs

Task Description-1
Fixing Syntax Errors
Expected Output:
After fixing the syntax and running the program, the function successfully adds the two numbers provided as input. For example, when add(7, 5) is executed, the program prints 12. This confirms that the corrected function now works properly and produces the correct result without any errors.
Prompt:
I have a Python function add(a, b) that is missing a colon. Please identify the syntax error, correct the function, and explain the issue in simple words.
CODE :

s1.py > ...

```
# Error code:
def add(a, b)
    return a + b

# Modified code:
def add(a, b):  # Added missing colon
    return a + b
```

Explanation:
The issue happened because Python needs a colon at the end of every function definition to understand where the function starts. Since the colon was missing, Python showed a syntax error and stopped running the program. The AI quickly identified this problem and added the missing colon, which solved the issue. Once this small change was made, the function executed normally and returned the correct sum of the numbers.

Task Description-2
Debugging Logic Errors in Loops
Expected Output:
When the corrected function is called with count_down(5), it prints the numbers 5, 4, 3, 2, 1, 0 one by one and then stops. The loop ends naturally once n becomes -1, which means the infinite loop problem is resolved and the program finishes properly.
Prompt:
I have a Python loop that is running infinitely because of a logic mistake. Please identify the problem, fix the loop, and explain the issue in simple words.
CODE :

s1.py > count_down

```python
# --- ERROR CODE ---
# Original buggy version: increments instead of decrementing
# This creates an infinite loop
def count_down(n):
    while n >= 0:
        print(n)
        n += 1  # BUG: This increments instead of decrementing
# --- CORRECTED VERSION ---
# Corrected version: decrements n instead of incrementing
# This properly counts down from n to 0
def count_down(n):
    while n >= 0:
        print(n)
        n += 1
        n -= 1  # Changed from n += 1 to n -= 1
```

Explanation:
The infinite loop happened because the value of n was increasing instead of decreasing, so the condition n >= 0 always stayed true. The AI noticed this mistake and replaced n += 1 with n -= 1, allowing the value of n to move closer to zero on each loop. Once n goes below zero, the condition becomes false and the loop stops.

Task Description-3
Handling Runtime Errors (Division by Zero)
Expected Output:
After adding the try-except block, the program no longer crashes when b is zero. Instead of stopping execution, it prints a friendly error message saying Error: Cannot divide by zero. This shows that the function now handles invalid input safely and continues running without breaking the program.
Prompt:
I have a Python function that crashes when dividing by zero. Please identify the runtime error, add try-except to handle it safely, and explain the solution in simple words.
CODE :

s1.py > ...

```python
def divide(a, b):
    return a / b
print(divide(10, 0))
# Modified code with error handling
def divide_safe(a, b):
    if b == 0:
        return "Error: Division by zero"
    return a / b
print(divide_safe(10, 0))
print(divide_safe(10, 2))
```

Explanation:
Earlier, the program failed because Python throws a ZeroDivisionError whenever we try to divide by zero. The AI fixed this by wrapping the division inside a try block and catching the problem using except. When the error occurs, the program now shows a clear message instead of crashing.

Task Description-4
Debugging Class Definition Errors
Expected Output:
After fixing the constructor, the class works correctly and allows us to create a Rectangle object. When we print the values, it shows 5 for length and 3 for width. This confirms that the object properties are now properly initialized and accessible.
Prompt:
I have a Python class where the constructor is missing self. Please identify the problem, fix the class definition, and explain why self is needed in simple words.
CODE :

s1.py > check_discount

```python
def check_discount(age, is_member):
    if age >= 60:
        print("Senior discount applied")
        if is_member:
            print("Additional member discount applied")
    else:
        if is_member:
            print("Member discount applied")
        else:
            print("No discount available")
check_discount(65, True)
check_discount(45, True)
check_discount(30, False)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS pwsh

```
PS C:\AIAC>
& C:\Users\ajayr\AppData\Local\Programs\Python\Python313\python.exe c:/A
Senior discount applied
Additional member discount applied
Member discount applied
No discount available
```

Explanation:
The problem happened because Python automatically sends the current object to the constructor, and this object must be received using the self parameter. Without self, Python does not know where to store the values of length and width, which leads to errors. The AI fixed this by adding self as the first parameter in __init__() and using it to assign instance variables.

Task Description-5
Resolving Index Errors in Lists
Expected Output:
After applying safe access methods, the program no longer crashes. Instead of stopping with an error, it prints a friendly message such as "Index is out of range." This shows that the list access is now handled safely, and the program continues running even when an invalid index is requested.
Prompt:
My Python program crashes when accessing a list index that doesn't exist. Please identify the error, fix the code using safe list access (length checking or try-except), and explain the solution in simple words.
CODE AND OUTPUT:

s1.py > ...

```python
numbers = [1, 2, 3]
# Access the third element (index 2) to avoid IndexError
try:
    print(numbers[2])
except IndexError:
    print("Index out of range")

```

Explanation:
The error occurred because the program tried to access a position in the list that doesn't exist. Since the list contains only three elements, asking for index 5 causes Python to raise an IndexError. The AI fixed this by adding either a try-except block or a length check before accessing the list.