

2303A51602

BATCH-25

ASSIGNMENT-7.3

Task 1: Fixing Syntax Errors

Scenario

You are reviewing a Python program where a basic function definition contains a syntax error.

Requirements

- Provide a Python function `add(a, b)` with a missing colon
- Use an AI tool to detect the syntax error
- Allow AI to correct the function definition
- Observe how AI explains the syntax issue

Expected Output

- Corrected function with proper syntax
- Syntax error resolved successfully
- AI-generated explanation of the fix

```
def add(a, b):
    return a + b
print(add(3, 5))

PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python31
3/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 7.3.py"
8
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

Task 2: Debugging Logic Errors in Loops

Scenario

You are debugging a loop that runs infinitely due to a logical mistake.

Requirements

- Provide a loop with an increment or decrement error
- Use AI to identify the cause of infinite iteration
- Let AI fix the loop logic
- Analyze the corrected loop behavior

Expected Output

- Infinite loop issue resolved
- Correct increment/decrement logic applied
- AI explanation of the logic error

The screenshot shows a dark-themed instance of Visual Studio Code. In the center, there is a code editor window displaying a Python file named `ass 7.3.py`. The code contains a buggy loop where `i` is never incremented, resulting in an infinite loop. A corrected version of the loop is shown below it. Below the code editor is a terminal window showing the execution of the script, which prints integers from 1 to 5. The status bar at the bottom indicates the path `C:\Users\parva\OneDrive\Desktop\AI Assted>`, line 10, column 10, and other terminal settings.

```
6
7     #Buggy loop (infinite loop due to missing increment)
8     #i = 1
9     #while i <= 5:
10    #print(i)
11    # i is never incremented + infinite loop"""
12
13
14    # Fixed loop (logic corrected)
15    i = 1
16    while i <= 5:
17        print(i)
18        i += 1
19    # increment added to stop infinite iteration
20
21
>> 2
>> 3
>> 4
>> 5
>>
1
2
3
4
5
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

Task 3: Handling Runtime Errors (Division by Zero)

Scenario

A Python function crashes during execution due to a division by zero error.

Requirements

- Provide a function that performs division without validation
- Use AI to identify the runtime error
- Let AI add try-except blocks for safe execution
- Review AI's error-handling approach

Expected Output

- Function executes safely without crashing
- Division by zero handled using try-except
- Clear AI-generated explanation of runtime error handling

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a file named 'ass 7.3.py'. The code in the editor is as follows:

```
20
21 # Buggy function (no validation → runtime error)
22 def divide(a, b):
23     return a / b
24
25 # This will crash if b = 0
26 # print(divide(10, 0))
27
28
29 # Fixed function using try-except (safe execution)
30 def safe_divide(a, b):
31     try:
32         return a / b
33     except ZeroDivisionError:
34         return "Error: Division by zero is not allowed"
35
36
37 # Testing
38 print(safe_divide(10, 2))
39 print(safe_divide(10, 0))
40
```

In the terminal tab, the output is:

```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 5.0
>> Error: Division by zero is not allowed
>>
```

The status bar at the bottom shows: Ln 40, Col 1 | Spaces: 4 | UTF-8 | CRLF | {} Python

Task 4: Debugging Class Definition Errors

Scenario

You are given a faulty Python class where the constructor is incorrectly defined.

Requirements

- Provide a class definition with missing self-parameter
- Use AI to identify the issue in the `__init__()` method
- Allow AI to correct the class definition
- Understand why `self` is required

Expected Output

- Corrected `__init__()` method
- Proper use of `self` in class definition
- AI explanation of object-oriented error

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** AI Assisted.
- Explorer Bar:** Shows AI ASSISTED, OUTLINE, and TIMELINE.
- Code Editor:** A Python file named "ass 7.3.py" is open. The code contains two classes: "Student". The first version is faulty (missing self parameter), and the second is corrected. A print statement at the bottom outputs "Mouna 20".
- Terminal:** Shows a PowerShell session with the command "PS C:\Users\parva\OneDrive\Desktop\AI Assisted> Mouna 20".
- Status Bar:** Shows Ln 58, Col 1, Spaces 4, UTF-8, CRLF.

Task 5: Resolving Index Errors in Lists

Scenario

A program crashes when accessing an invalid index in a list.

Requirements

- Provide code that accesses an out-of-range list index
- Use AI to identify the Index Error
- Let AI suggest safe access methods
- Apply bounds checking or exception handling

Expected Output

- Index error resolved
- Safe list access logic implemented

The screenshot shows the GitHub Copilot AI Assistant integrated into the Visual Studio Code (VS Code) interface. The main area displays a Python file named `ass 7.3.py` with code related to handling list index errors. A sidebar on the right provides AI assistance, including a prompt to identify and fix syntax errors and a message about reaching a monthly message quota.

```
# Buggy code (IndexError: list index out of range)
numbers = [10, 20, 30]
print(numbers[5]) # invalid index

# Fixed code using exception handling (safe access)
numbers = [10, 20, 30]
try:
    print(numbers[5])
except IndexError:
    print("Error: Index out of range")

# Fixed code using bounds checking
index = 2
if index < len(numbers):
    print(numbers[index])
else:
    print("Error: Index out of range")
```

TERMINAL

```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> Error: Index out of range
>> 30
>> 
```