

2303A51556

Batch:25

Task 1: AI-Assisted Syntax and Code Quality Review

Scenario

You join a development team and are asked to review a junior developer's Python script that fails to run correctly due to basic coding mistakes. Before deployment, the code must be corrected and standardized.

Task Description

You are given a Python script containing:

- Syntax errors
- Indentation issues
- Incorrect variable names
- Faulty function calls

Use an AI tool (GitHub Copilot / Cursor AI) to:

- Identify all syntactic and structural errors
- Correct them systematically
- Generate an explanation of each fix made

Expected Outcome

- Fully corrected and executable Python code
- AI-generated explanation describing:
 - Syntax fixes
 - Naming corrections
 - Structural improvements
- Clean, readable version of the script

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

- File Explorer:** Shows a folder structure including 'new.py'.
- Run and Debug:** Shows a configuration for 'Python'.
- VARIABLES:** Shows variables from the current scope.
- WATCH:** Shows watched variables.
- CALL STACK:** Shows the call stack.
- PROBLEMS:** Shows no problems.
- OUTPUT:** Shows the terminal output:

 - Line 1: PS C:\Users\chiranjit\Downloads\Devops & 'C:\Users\chiranjit\AppData\Local\Microsoft\Windows\Python\python.11.exe' 'c:\users\chiranjit\vscode\extensions\ms-python.python.debug-2023.10.0-win32-x64\pinned\linter\debugger' '198411' '--> 'C:\Users\chiranjit\Downloads\Devops\new.py'
 - Line 2: Enter for file is £0.00
 - Line 3: Hello, Soumya!
 - Line 4: Sum: 60
 - Line 5: PS C:\Users\chiranjit\Downloads\Devops>

- BREAKPOINTS:** Shows breakpoints for 'User Uncaught Exceptions'.
- Python Debugger:** Shows the Python file (Devops).

Task 2: Performance-Oriented Code Review

Scenario

A data processing function works correctly but is inefficient and slows down the system when large datasets are used.

Task Description

You are provided with a function that identifies duplicate values in a list using inefficient nested loops.

Using AI-assisted code review:

- Analyze the logic for performance bottlenecks
 - Refactor the code for better time complexity
 - Preserve the correctness of the output

Ask the AI to explain:

- Why the original approach was inefficient
 - How the optimized version improves performance

Expected Outcome

- Optimized duplicate-detection logic (e.g., using sets or hash-based structures)
 - Improved time complexity
 - AI explanation of performance improvement
 - Clean, readable implementation

```

10 def find_duplicates_slow(data):
11     duplicates = []
12     for i in range(len(data)):
13         for j in range(i+1, len(data)):
14             if data[i] == data[j] and data[i] not in duplicates:
15                 duplicates.append(data[i])
16     return duplicates
17
18 def find_duplicates_fast(data):
19     seen = set()
20     duplicates = set()
21
22     for item in data:
23         if item in seen:
24             duplicates.add(item)
25         else:
26             seen.add(item)
27
28     return list(duplicates)
29
30
31 def explanation():
32     print("AI Code Review Explanation:\n")
33
34     print("1. Why the original approach was inefficient:")
35     print("   It used nested loops, resulting in O(n^2) time complexity.")
36     print("2. Each element was compared with every other element.")
37     print("3. Checking duplicates in a list also takes extra time.\n")
38
39     print("2. Why the optimized version improves performance:")
40     print("1. Uses a set (hash-based structure) for constant-time lookup O(1).")
41     print("2. Traverses the list only once, giving O(n) time complexity.")
42     print("3. Significantly faster for large datasets.\n")
43
44 if __name__ == "__main__":
45     data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 1]
46
47     print("Duplicates (Slow Method):", find_duplicates_slow(data))
48     print("Duplicates (Optimized Method):", find_duplicates_fast(data))
49
50     print()
51     explanation()

```

```

10 def find_duplicates_fast(data):
11     duplicates = set()
12
13     for item in data:
14         if item in seen:
15             duplicates.add(item)
16         else:
17             seen.add(item)
18
19     return list(duplicates)
20
21
22 def explanation():
23     print("AI Code Review Explanation:\n")
24
25     print("1. Why the original approach was inefficient:")
26     print("   It used nested loops, resulting in O(n^2) time complexity.")
27     print("2. Each element was compared with every other element.")
28     print("3. Checking duplicates in a list also takes extra time.\n")
29
30     print("2. Why the optimized version improves performance:")
31     print("1. Uses a set (hash-based structure) for constant-time lookup O(1).")
32     print("2. Traverses the list only once, giving O(n) time complexity.")
33     print("3. Significantly faster for large datasets.\n")
34
35 if __name__ == "__main__":
36     data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 1]
37
38     print("Duplicates (Slow Method):", find_duplicates_slow(data))
39     print("Duplicates (Optimized Method):", find_duplicates_fast(data))
40
41     print()
42     explanation()
43
44

```

Task 3: Readability and Maintainability Refactoring

Scenario

A working script exists in a project, but it is difficult to understand due to poor naming, formatting, and structure. The team wants it rewritten for long-term maintainability.

Task Description

You are given a poorly structured Python function with:

- Cryptic function names
- Poor indentation

- Unclear variable naming
- No documentation

Use AI-assisted review to:

- Refactor the code for clarity
- Apply PEP 8 formatting standards
- Improve naming conventions
- Add meaningful documentation

Expected Outcome

- Clean, well-structured code
- Descriptive function and variable names
- Proper indentation and formatting
- Docstrings explaining the function purpose
- AI explanation of readability improvements

```

RUN AND DEBUG  Python...  new.py
C:\Users\Chinna>Downloads> Python> ...
1 def calculate_student_average(marks):
2     if not isinstance(marks, list) or len(marks) == 0:
3         raise ValueError("Marks must be a non-empty list.")
4
5     total_marks = 0
6     for score in marks:
7         if not isinstance(score, (int, float)):
8             raise ValueError("All marks must be numbers.")
9         total_marks += score
10
11     average = total_marks / len(marks)
12     return average
13
14
15 def ai_explanation():
16     print("AI Refactoring Explanation:\n")
17
18     print("Readability Improvements:")
19     print("1. Renamed function to 'calculate_student_average' for clarity.")
20     print("2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.")
21     print("3. Applied proper indentation and spacing following PEP 8 standards.")
22
23     print("\nMaintainability Improvements:")
24     print("1. Added a clear docstring explaining purpose, arguments, and return value.")
25     print("2. Added input validation and error handling.")
26     print("3. Structured the code into a reusable function.")
27
28     print("\nOverall Result:")
29     print("The code is now easier to read, understand, modify, and maintain.")

CALLSTACK
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
AI Refactoring Explainer:
Readability Improvements:
1. Renamed function to 'calculate_student_average' for clarity.
2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.
3. Applied proper indentation and spacing following PEP 8 standards.

Maintainability Improvements:
1. Added a clear docstring explaining purpose, arguments, and return value.
2. Added input validation and error handling.
3. Structured the code into a reusable function.

Breakpoints:
Raised Exceptions
Uncaught Exceptions
User Uncaught Exceptions
Python Debugger: Python File (DevOps)

PS C:\Users\Chinna>cd "C:\Users\Chinna\Downloads\DevOps"; & "C:\Users\Chinna\Downloads\DevOps\Python\venv\Scripts\python3.10.exe" "C:\Users\Chinna\Downloads\DevOps\new.py"
Average Marks: 86.6

```

```

File Edit Selection View Go Run Terminal Help ← → C DevOps
RUN AND DEBUG Python new.py X
C:\Users\Chinmaya\Downloads> new.py
14
15     def ai_explanation():
16         print("AI Refactoring Explanation:")
17
18         print("1. Renamed function to 'calculate_student_average' for clarity.")
19         print("2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.")
20         print("3. Applied proper indentation and spacing following PEP 8 standards.")
21
22
23         print("4. Maintainability Improvements:")
24         print("1. Added a clear docstring explaining purpose, arguments, and return value.")
25         print("2. Added input validation and error handling.")
26         print("3. Structured the code into a reusable function.")
27
28         print("Overall Result:")
29         print("The code is now easier to read, understand, modify, and maintain.")
30
31
32 if __name__ == "__main__":
33     student_marks = [85, 90, 78, 92, 88]
34
35     try:
36         avg = calculate_student_average(student_marks)
37         print("Average Marks:", avg)
38     except Exception as e:
39         print(str(e))
40
41     print()
42     ai_explanation()
43

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\Chinmaya\Downloads\DevOps> cd c:\Users\Chinmaya\Downloads\DevOps & "C:\Users\Chinmaya\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "c:\Users\Chinmaya\vscodeextensions\ms-python.debug-2023.10.0-win32-x64\kunstd\lib\"
1. Renamed function to 'calculate_student_average' for clarity.
2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.
3. Applied proper indentation and spacing following PEP 8 standards.

Maintainability Improvements:
1. Added a clear docstring explaining purpose, arguments, and return value.
2. Added input validation and error handling.
3. Structured the code into a reusable function.

Overall Result:
The code is now easier to read, understand, modify, and maintain.
PS C:\Users\Chinmaya\Downloads\DevOps>

```

BREAKPOINTS

- Raised Exceptions
- Uncaught Exceptions
- User Breakpoint Exceptions

Task 4: Secure Coding and Reliability Review

Scenario

A backend function retrieves user data from a database but has security vulnerabilities and poor error handling, making it unsafe for production deployment.

Task Description

You are given a Python script that:

- Uses unsafe SQL query construction
- Has no input validation
- Lacks exception handling

Use AI tools to:

- Identify security vulnerabilities
- Refactor the code using safe coding practices
- Add proper exception handling
- Improve robustness and reliability

Expected Outcome

- Secure SQL queries using parameterized statements
- Input validation logic
- Try-except blocks for runtime safety
- AI-generated explanation of security improvements

- Production-ready code structure give code for this remove comments

Task 5: AI-Based Automated Code Review Report

Scenario

Your team uses AI tools to perform automated preliminary code reviews before human review, to improve code quality and consistency across projects.

Task Description

You are provided with a poorly written Python script.

Using AI-assisted review:

- Generate a structured code review report that evaluates:

- #### o Code readability

- ### o Naming conventions

- #### o Formatting and style consistency

- ### o Error handling

- #### o Documentation quality

- #### o Maintainability

- The task is not ju

issues.

Expect

• AI-generated reviews

- o Identified quality issues
 - o Risk areas
 - o Code smell detection
 - o Improvement suggestions
 - Optional improved version of the code
 - Demonstration of AI as a code reviewer, not just a code

Generator

The screenshot shows the Microsoft Visual Studio Code interface with a Python file open for review. The code defines a function to calculate division and then prints a detailed report of automated code review findings across various categories like Readability, Naming Conventions, and Error Handling.

```
def poorly_written_function(a, b):
    c = a / b
    print("Result:", c)
    return c

def Improved_Function(number1, number2):
    if not isinstance(number1, (int, float)) or not isinstance(number2, (int, float)):
        raise ValueError("Inputs must be numeric")
    if number2 == 0:
        raise ValueError("Division by zero is not allowed")
    result = number1 / number2
    return result

def ai_code_review_report():
    print("AI Automated Code Review Report!")

    print("1. Code Readability Issues:")
    print("- Function name was long and not descriptive.")
    print("- Variable names (a, b, c) did not indicate purpose.")
    print("- Logic and output were mixed together.")

    print("2. Naming Convention Issues:")
    print("- Did not follow descriptive naming standards.")
    print("- Improved version uses meaningful names like number1 and result.")

    print("3. Formatting and Style Consistency:")
    print("- No consistent spacing or structure.")
    print("- Improved version follows standard Python formatting.")

    print("4. Error Handling Problems:")
    print("- No validation for invalid inputs.")
    print("- No protection against division by zero.")
    print("- Improved version adds input validation and exceptions.")

    print("5. Documentation Quality:")
    print("Improved Result: 5.0")

AI Automated Code Review Report

1. Code Readability Issues:
- Function name was long and not descriptive.
- Variable names (a, b, c) did not indicate purpose.
- Logic and output were mixed together.

2. Naming Convention Issues:
- Did not follow descriptive naming standards.
- Improved version uses meaningful names like number1 and result.

3. Formatting and Style Consistency:
- No consistent spacing or structure.
- Improved version follows standard Python formatting.

4. Error Handling Problems:
- No validation for invalid inputs.
- No protection against division by zero.
- Improved version adds input validation and exceptions.

5. Documentation Quality:
Improved Result: 5.0
```

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a file named `new.py` in the current workspace.
- Code Editor:** Displays a Python script with various print statements outputting code review results. The code includes sections for documentation quality, maintainability risks, and code smells, along with improvement suggestions and error handling.
- Output Panel:** Shows the command line output of running the script, which includes the improved function results and the final message: "Improved version is safer, more readable, and maintainable."
- Terminal:** Shows the command used to run the script: `python new.py`.
- Bottom Status Bar:** Shows the file name `new.py`, line count (L+25), column count (Col 1), and other status indicators.

The screenshot shows the Visual Studio Code interface with a Python file named `new.py` open. The code is a script for generating a code review report. It includes sections for code smells, improvement suggestions, and a main function that prints the result. The terminal shows the execution of the script and its output.

```
C:\> users> chmier> Downloads > newpy > a_code_review_report
C:\> DevOps

File Edit Selection View Go Run Terminal Help ← → C:\> DevOps

RUN AND DEBUG Python - X
VARIABLES
W WATCH
CALL STACK

C:\> users> chmier> Downloads > newpy > a_code_review_report
1 def a_code_review_report():
2     print("No documentation or explanation of function behavior.")
3     print("- Suggested adding docstrings for production code.")
4
5     print("Magic variables detected!")
6     print("- Hard to extend due to poor naming and structure.")
7     print("- Mixing computation with printing reduces reusability.")
8
9     print("Code Smells Detected:")
10    print("- Magic variables")
11    print("- Lack of validation")
12    print("- Tight coupling between logic and UI output")
13
14    print("Improvement Suggestions:")
15    print("- Use descriptive names")
16    print("- Separate logic from presentation")
17    print("- Add validation and error handling")
18    print("- Follow consistent formatting standards")
19
20    print("Result")
21    print("Improved version is safer, more readable, and maintainable.")
22
23 if __name__ == "__main__":
24     try:
25         print("Improved Result:", improved_function(20, 2))
26     except Exception as e:
27         print("Error:", e)
28
29     print()
30     a_code_review_report()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ - | ×
powershell
Python
Python Debug

PS C:\Users\chmier\Downloads\DevOps> cd c:\Users\chmier\Downloads\DevOps; & "C:\Users\chmier\AppData\Local\Microsoft\WindowsApps\python3.10.exe" "c:\Users\chmier\vscode\extensions\ms-python.python\2025.10.0\python\bin\python.exe" "c:\Users\chmier\vscode\extensions\ms-python.python\2025.10.0\python\bin\runpy.py" "a_code_review_report"
Improved Result: 5.0

7. Code Smells Detected:
- Magic variables.
- Lack of validation.
- Tight coupling between logic and UI output.

8. Improvement Suggestions:
- Use descriptive names.
- Separate logic from presentation.
- Add validation and error handling.
- Follow consistent formatting standards.

Result:
Improved version is safer, more readable, and maintainable.
PS C:\Users\chmier\Downloads\DevOps>

BREACKPOINTS
Breakpoints
Uncaught Exceptions
User Uncaught Exceptions

Python Debugger: Python File (DevOps)

Lo 25, Col 1 Spaces 4 UTF-8 CR/LF Python Python 3.10
```