

2303A51504

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 a VS Code interface with the following details:

- File Explorer:** Shows a folder structure including 'new.py'.
- Terminal:** Displays the output of running the script: "C:\Users\Chimari> python > -> new.py". The script defines three functions: calculate_discount, greet_username, and sum_numbers. It also includes a main block with a try-except block for handling exceptions.
- Code Editor:** The 'new.py' file is open, showing the implementation of the functions.
- Output:** Shows the result of the script execution: "Discount for <price> is <discount>" followed by the greeting "Hello, Swami!" and the sum of numbers [10, 20, 30].
- Breakpoints:** A breakpoint is set at line 41 of the script.
- Python Debugger:** A status bar at the bottom indicates "Python Debugger 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

The screenshot shows the VS Code interface with the Python extension installed. The code editor displays a file named `new.py` containing two functions: `find_duplicates_slow(data)` and `find_duplicates_fast(data)`. The `slow` version uses nested loops to compare every element against every other. The `fast` version uses sets to store seen items and add new ones, comparing each item against the set. An `explanation()` function provides a detailed code review explanation. The terminal shows the command to run the script and its output, which compares the execution times of both methods.

```
C:\Users\Chinmaya\Downloads> python new.py
[1, 2, 3, 4, 2, 5, 6, 3, 7, 1]
Duplicates (Slow Method): [1, 2, 3]
Duplicates (Optimized Method): [1, 2, 3]

PS C:\Users\Chinmaya\Downloads> cd "C:\Users\Chinmaya\Downloads\Devops"; & "C:\Users\Chinmaya\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "C:\Users\Chinmaya\vscodeextensions\ms-python.debug-2025.10.0-win32-x64\bundle\lib\libps
debug\launcher" "5764" "..."\C:\Users\Chinmaya\Downloads\new.py
Duplicates (Slow Method): [1, 2, 3]
Duplicates (Optimized Method): [1, 2, 3]
```

This screenshot shows the same `new.py` file in VS Code, but with a different configuration. The terminal output is identical to the first screenshot, indicating the same results for both methods. The configuration difference is likely related to the Python environment or debugger settings.

```
PS C:\Users\Chinmaya\Downloads> cd "C:\Users\Chinmaya\Downloads\Devops"; & "C:\Users\Chinmaya\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "C:\Users\Chinmaya\vscodeextensions\ms-python.debug-2025.10.0-win32-x64\bundle\lib\libps
debug\launcher" "5764" "..."\C:\Users\Chinmaya\Downloads\new.py
Duplicates (Slow Method): [1, 2, 3]
Duplicates (Optimized Method): [1, 2, 3]

PS C:\Users\Chinmaya\Downloads> cd "C:\Users\Chinmaya\Downloads\Devops"; & "C:\Users\Chinmaya\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "C:\Users\Chinmaya\vscodeextensions\ms-python.debug-2025.10.0-win32-x64\bundle\lib\libps
debug\launcher" "5764" "..."\C:\Users\Chinmaya\Downloads\new.py
Duplicates (Slow Method): [1, 2, 3]
Duplicates (Optimized Method): [1, 2, 3]
```

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...  VARIABLES  WATCH  CALLSTACK  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  SORTS
C:\Users\Chimer1\Downloads> cd newpy >...
newpy
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("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("\nOverall Result:")
29     print("The code is now easier to read, understand, modify, and maintain.\n")
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(f"Error: {e}")
40 
41 
42 PS C:\Users\Chimer1\Downloads\Devops> cd c:\Users\Chimer1\Downloads\Devops & & "C:\Users\Chimer1\AppData\Local\Microsoft\WindowsApps\Python3.11.exe" "c:\Users\Chimer1\.vscode\extensions\ms-python.python-2025.18.0\python\debugpy-2025.18.0-win32-win32\bundles\dlls\pythonw.exe" <C:\users\chimer1\Downloads\newpy.py
Average Marks: 86.0
AI Refactoring Explanation:
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.

```

```

14     def ai_explanation():
15         print("AI Refactoring Explanation:\n")
16
17         print("Readability Improvements:")
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         print("Maintainability Improvements:")
23         print("1. Added a clear docstring explaining purpose, arguments, and return value.")
24         print("2. Added input validation and error handling.")
25         print("3. Structured the code into a reusable function.")
26
27         print("\nOverall Result:")
28
29         print("The code is now easier to read, understand, modify, and maintain.\n")
30
31
32 if __name__ == "__main__":
33     student_marks = [85, 98, 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("Error:", e)
40
41     print()
42     ai_explanation()
43

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Chinmay\Downloads\Devops> cd c:\Users\Chinmay\Downloads\Devops & <C:\Users\Chinmay\Downloads\Devops> & <C:\Users\Chinmay\Downloads\Devops> & <C:\Users\Chinmay\Downloads\Devops> & <C:\Users\Chinmay\Downloads\Devops>

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.

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

The screenshot shows a Python code editor interface with a script named 'new.py'. The code is a simple database query handler. Below the code, there is an AI-generated security review report. The report highlights several improvements and vulnerabilities:

```

C:\Users\Chinmari>cd "C:\Users\Chinmari\Downloads\Devops"
C:\Users\Chinmari>python new.py
  File "new.py", line 1
    import sqlite
    ^
SyntaxError: invalid syntax

AI Security Review Report:
  File: new.py
  Line: 1
  Problem: SyntaxError: invalid syntax
  Fix: Add a blank line before the import statement.

  File: new.py
  Line: 10
  Problem: Replaced unsafe SQL string concatenation with parameterized query.
  Fix: Replace the unsafe query with a parameterized one.

  File: new.py
  Line: 17
  Problem: Added input validation to ensure user_id is a positive integer.
  Fix: Add a check to ensure user_id is a positive integer.

  File: new.py
  Line: 35
  Problem: Prevented SQL injection vulnerabilities.
  Fix: Use parameterized queries to prevent SQL injection.

```

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 just to fix the code, but to analyze and report on quality issues.

Expected Outcome

- AI-generated review report including:

- o Identified quality issues

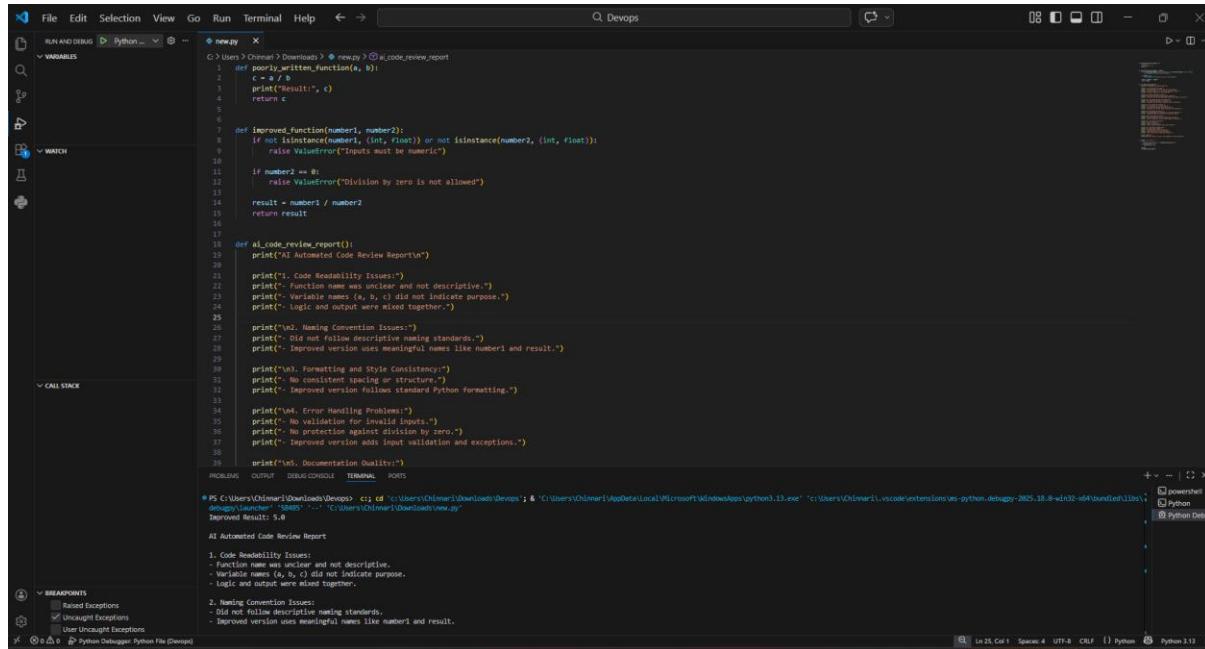
- o Risk areas

- 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



```
G > Users \ Chinmari \ Downloads > new.py (ai_code_review_report)
def improved_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 unclear 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("- No documentation or explanation of function behavior.")
    print("- Suggested adding docstrings for production code.")

    print("6. Maintainability Risks:")
    print("- Hard to extend due to poor naming and structure.")
    print("- Mixing computation with printing reduces reusability.")

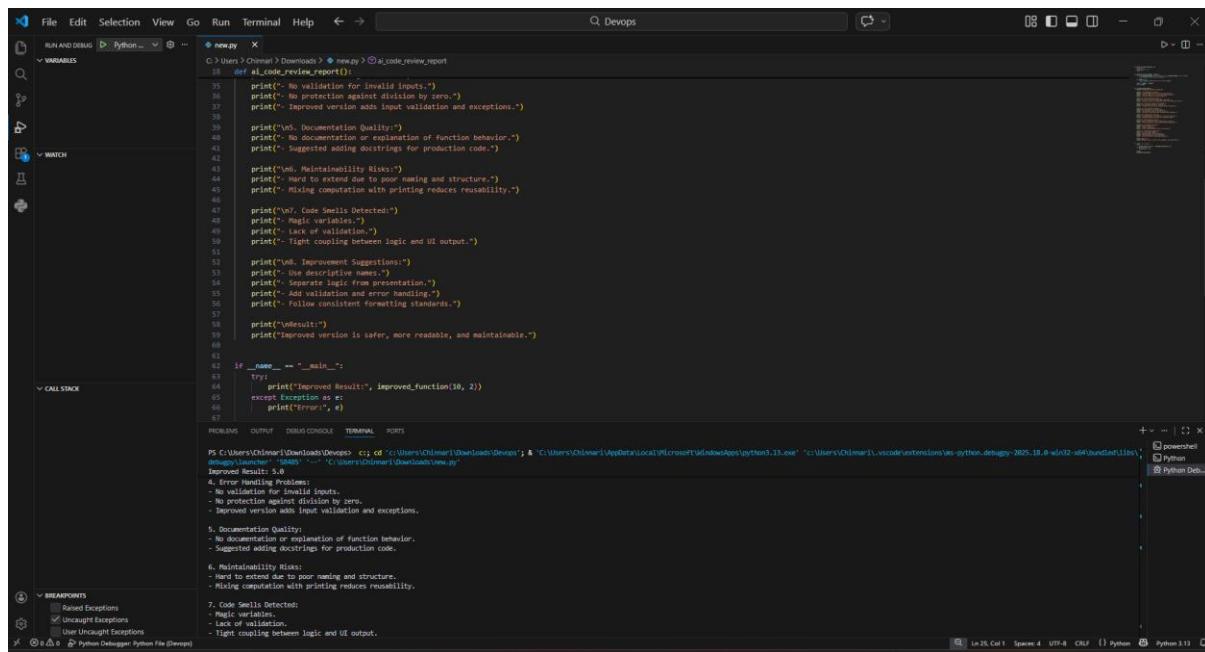
    print("7. Code Smells Detected:")
    print("- Magic variables.")
    print("- Lack of validation.")
    print("- Tight coupling between logic and UI output.")

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

    print("9. Result:")
    print("Improved version is safer, more readable, and maintainable.")

if __name__ == "__main__":
    try:
        print("Improved Result:", improved_function(10, 2))
    except Exception as e:
        print(str(e))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```



```
G > Users \ Chinmari \ Downloads > new.py (ai_code_review_report)
def improved_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 unclear 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("- No documentation or explanation of function behavior.")
    print("- Suggested adding docstrings for production code.")

    print("6. Maintainability Risks:")
    print("- Hard to extend due to poor naming and structure.")
    print("- Mixing computation with printing reduces reusability.")

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

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

    print("9. Result:")
    print("Improved version is safer, more readable, and maintainable.")

if __name__ == "__main__":
    try:
        print("Improved Result:", improved_function(10, 2))
    except Exception as e:
        print(str(e))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

The screenshot shows the Microsoft Visual Studio Code interface with the Python extension installed. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and DevOps. The left sidebar has sections for RUN & DEBUG, VARIABLES, WATCH, and CALL STACK. The main editor area contains a Python script named `ai_code_review_report.py`. The script uses the `ai_code_review_report()` function to print various code review findings. The bottom status bar shows the file path as `C:\Users\Chimarr\Downloads\Devops\ai_code_review_report.py`, the line number as 1, and the column number as 1.

```
C:\Users\Chimarr\Downloads\Devops> ai_code_review_report()
  File "C:\Users\Chimarr\Downloads\Devops\ai_code_review_report.py", line 1
    print("No documentation or explanation of function behavior.")
    ^
[...]
  File "C:\Users\Chimarr\Downloads\Devops\ai_code_review_report.py", line 1
    print("Improved version is safer, more readable, and maintainable.")

  +--> [Run] Python Debug (Python File (Devops))

  +--> [Breakpoints]
      Raised Exceptions
      Uncought Exceptions
      User Uncought Exceptions

  +--> [Python Debugger] Python File (Devops)

  +--> [Terminal]
      PS C:\Users\Chimarr\Downloads\Devops> ai_code_review_report()
      Improved Result: 5.0

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

      8. Improvement Suggestions:
      - Use descriptive names.
      - Simplify code representation.
      - Add validation and error handling.
      - Follow consistent formatting standards.

      Result:
      Improved version is safer, more readable, and maintainable.

  +--> [PROBLEMS]
      PS C:\Users\Chimarr\Downloads\Devops>
```