

AI-Ass-10.4_1312

Name:T.Shivani

Ht.no:2303A51312

Batch:05

Task 1: AI-Assisted Syntax and Code Quality Review

Prompt:#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:

#o Syntax fixes

Naming corrections

Structural improvements

Clean, readable version of the script

Code:

Error code

```
def AddNumbers(a,b)
```

```
result = a + B
```

```
print("Sum is",result)
```

```
Addnumbers(5, 10)
```

The screenshot shows the Visual Studio Code interface. In the top right corner, there is an 'AI-assisted' button. The Explorer sidebar on the left lists several Python files, including 'Ass-10.4.py'. The main code editor window displays the following Python code:

```
def AddNumbers(a,b)
result = a + B
print("Sum is",result)

Addnumbers(5, 10)
```

The code editor has a light theme with dark syntax highlighting. Below the code editor is a 'Postman Console' tab, which is currently inactive. At the bottom of the screen is a taskbar with various icons.

In the terminal window at the bottom, there is an error message:

```
PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted> cd 'C:\Users\Shivani\OneDrive\Desktop\AI-assisted'; & 'c:\Users\Shivani\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani\OneDrive\Desktop\AI-assisted\Ass-10.4.py'
File "C:\Users\Shivani\OneDrive\Desktop\AI-assisted\Ass-10.4.py", line 2, in <module>
    def AddNumbers(a,b)
    ^
SyntaxError: expected ':'
```

Correct code:

```
def add_numbers(a, b):
```

```
    result = a + b
```

```
    print("Sum is", result)
```

```
add_numbers(5, 10)
```

The screenshot shows a Windows desktop environment with VS Code open. The terminal window displays the following Python code and its execution:

```

6  #Standardized.
7  #Task Description
8  #You are given a Python script containing:
9  #
10 #Syntax errors
11 #Formatting issues
12 #Incorrect variable names
13 #Faulty function calls
14 #Use an AI tool (GitHub Copilot / Cursor AI) to:
15 # Identify all syntactic and structural errors
16 # Correct them systematically
17 # Generate an explanation of each fix made
18 #Expected Outcome
19 # Fully corrected and executable Python code
20 #A detailed explanation describing:
21 # Syntax fixes
22 # Naming corrections
23 # Structural improvements
24 def add_numbers(a, b):
25     result = a + b
26     print("Sum is", result)
27
28
29 add_numbers(5, 10)

```

The terminal also shows the command to run the script and its output:

```

PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted> & "c:\Users\Shivani\AppData\Local\Python\pythoncore-3.14-64\python.exe" "c:\Users\Shivani\OneDrive\Desktop\AI-assisted\ms_python.debugpy-2025.18.0-win32-x64\build\libs\debug\launcher" "60362" "-l" "C:\Users\Shivani\OneDrive\Desktop\AI-assisted\Ass-10.4.py"
PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted>

```

The status bar at the bottom right shows: Python 3.14.2, ENG IN, and 1008 05-02-2026.

Prompt: Task 2: Performance-Oriented Code Review

#Scenario

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
```

Code:

Original Inefficient Code (Nested Loops)

```
def find_duplicates(nums):
```

```
    duplicates = []
```

```
    for i in range(len(nums)):
```

```
        for j in range(i + 1, len(nums)):
```

```
            if nums[i] == nums[j] and nums[i] not in duplicates:
```

```
                duplicates.append(nums[i])
```

```
    return duplicates
```

#example usage

```
numbers = [1, 2, 3, 2, 4, 5, 1]
```

```
print(find_duplicates(numbers))
```

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files such as coding.py, lab-02.py, Ass-8.4.py, Ass-9.3.py, and Ass-10.4.py. The main editor area displays the Python code for finding duplicates. The terminal at the bottom shows command-line output related to the AI-assisted code review process.

```
File Edit Selection View Go Run Terminal Help ⇨ → AI-assisted

OPEN FOLDERS
EXPLORER
AI-ASSISTED
RECENTS
DATA
LOGS
OUTPUT
TERMINAL
PORTS
POSTMAN CONSOLE
PROBLEMS
DEBUG CONSOLE
AI-ASSISTED
RECOMMENDED
DATA
LOGS
OUTPUT
TERMINAL
PORTS
POSTMAN CONSOLE
PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
POSTMAN CONSOLE
PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted> & 'c:\Users\Shivani\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani\OneDrive\Desktop\AI-assisted\Ass-10.4.py'
[1, 2]
PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted>
```

The code in the editor is as follows:

```
def find_duplicates(nums):
    duplicates = []
    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            if nums[i] == nums[j] and nums[i] not in duplicates:
                duplicates.append(nums[i])
    return duplicates
```

Optimized Code (Using Set)

```
def find_duplicates(nums):
```

```
    seen = set()
```

```
    duplicates = set()
```

```
    for num in nums:
```

```
        if num in seen:
```

```
            duplicates.add(num)
```

```
        else:
```

```
            seen.add(num)
```

```
    return list(duplicates)
```

```
# Example usage
```

```
numbers = [1, 2, 3, 4, 2, 3, 5, 6, 1]
```

```
print("Duplicates are:", find_duplicates(numbers))
```

The screenshot shows the Visual Studio Code interface. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The title bar says "Q_Ai-assisted". The Explorer sidebar on the left lists various Python files and folders. The main editor area displays the optimized code for finding duplicates using sets. The terminal at the bottom shows the execution of the script, outputting the list of duplicates [1, 2].

```
48     # Preserve the correctness of the output
49     #Ask the AI to explain:
50     # Why the original approach was inefficient
51     # How the optimized version improves performance
52     #Expected Outcome
53     # Optimized duplicate-detection logic (e.g., using sets or hash-
54     #maps) instead of loops
55     # Improved time complexity
56     # AI explanation of performance improvement
57     # Clean, readable implementation
58
59     #Optimized Code (Using Set)
60     def find_duplicates(nums):
61         seen = set()
62         duplicates = set()
63
64         for num in nums:
65             if num in seen:
66                 duplicates.add(num)
67             else:
68                 seen.add(num)
69
70     return list(duplicates)
71
72     # Example usage
73 numbers = [1, 2, 3, 4, 2, 3, 5, 6, 1]
74 print("Duplicates are:", find_duplicates(numbers))
```

```
PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted> & 'c:\Users\Shivani\VScode\Local\Python\core-3.14-64\python.exe' 'c:\Users\Shivani\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bu-
D:\lib\lib\debug\launcher' -62771 & 'c:\Users\Shivani\OneDrive\Desktop\AI-assisted\Ass-10.4.py'
PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted> c: cd 'c:\Users\Shivani\OneDrive\Desktop\AI-assisted'; & 'c:\Users\Shivani\VScode\Local\Python\core-3.14-64\python.exe' 'c:\Users\Shivani\vscode\exten-
Duplicates are: [1, 2, 3]
PS C:\Users\Shivani\OneDrive\Desktop\AI-assisted> []
```

Prompt: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

Code:

Poorly Structured Code (Original)

```
def f(l):
```

```
    x=0
```

```
    for i in l:
```

```
        if i%2==0:
```

```
x=x+i
```

```
return x
```

```
#Example usage
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

```
print(f(numbers))
```

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

- File Explorer:** Shows files in the project, including `coding.py`, `lab-02.py`, `Ass-8.4.py`, `Ass-9.3.py`, and `Ass-10.4.py`. `Ass-10.4.py` is the active file.
- Editor:** Displays the code for `Ass-10.4.py`. The code contains a function `f` that iterates through a list of numbers and adds them to a variable `x`. The AI-assisted review suggests several improvements, such as better naming, documentation, and readability.
- Terminal:** Shows the command line output of a Python debugger session. It includes environment variables like `PS` and paths related to the AI-assisted extension.
- Status Bar:** Provides information about the current file (line 117, column 18), encoding (ENG IN), and date (05-02-2026).

```
def calculate_even_sum(numbers):
```

```
    """
```

Calculate the sum of all even numbers in a given list.

Args:

 numbers (list of int): List containing integer values.

Returns:

 int: Sum of even numbers in the list.

```
    """
```

```
even_sum = 0
```

```
for number in numbers:
```

```

if number % 2 == 0:

    even_sum += number

return even_sum

# Example usage

num_list = [1, 2, 3, 4, 5, 6]

print("Sum of even numbers:", calculate_even_sum(num_list))

```

The screenshot shows the Visual Studio Code interface. The left sidebar shows a file tree with various files like 'coding.py', 'lab-02.py', 'Ass-8.4.py', 'Ass-9.1.py', and 'Ass-10.4.py'. The main editor area contains the Python code provided above. The terminal at the bottom shows the command `python Ass-10.4.py` being run, followed by the output "Sum of even numbers: 12". The status bar at the bottom right indicates the date as 05-02-2026.

Prompt: 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

```

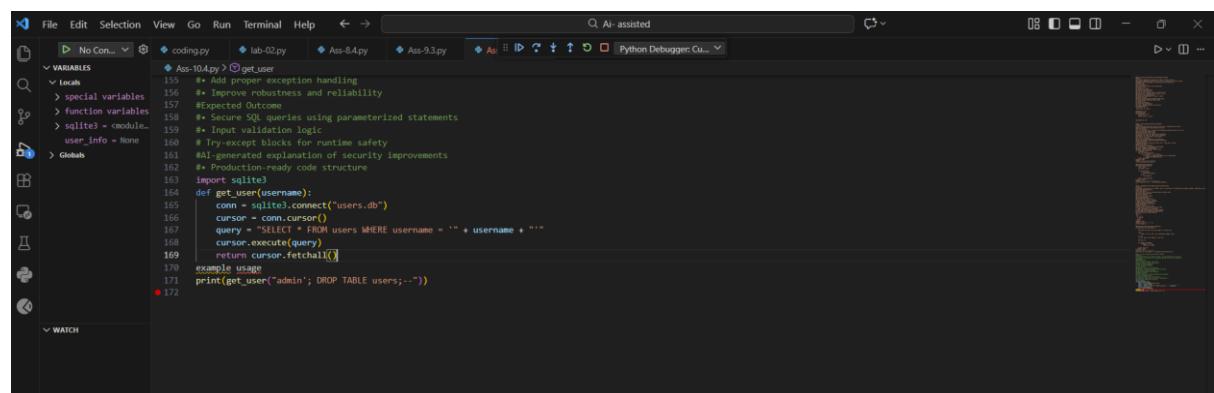
Code:

```

import sqlite3

def get_user(username):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE username = '" + username + "'"
    cursor.execute(query)
    return cursor.fetchall()

```



```

import sqlite3

def get_user(username):
    """
    """

```

Retrieve user details safely from the database.

.....

```
if not isinstance(username, str) or not username.strip():
```

```
    raise ValueError("Invalid username provided.")
```

try:

```
    with sqlite3.connect("users.db") as conn:
```

```
        cursor = conn.cursor()
```

```
        cursor.execute(
```

```
            "SELECT * FROM users WHERE username = ?",

```

```
            (username,))

```

```
)

```

```
    return cursor.fetchall()

```

except sqlite3.Error as error:

```
    print(f"Database error occurred: {error}")

```

```
return None
```

The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Find, and Run.
- Code Editor:** Displays Python code for a function named `get_user`. The code includes comments explaining the use of parameterized SQL statements and try-except blocks for runtime safety.
- Explorer:** Shows a tree view of the project structure. Files listed include `coding.py`, `lab-02.py`, `Ass-84.py`, `Ass-93.py`, `Ass-104.py` (the active file), `Ass-104.py ~ get_user`, `expected_outcome.py`, `secure_sql.py`, `input_validation.py`, `try_except.py`, `production_ready.py`, `log_keyword_analysis.py`, `README.md`, `sample_data.csv`, `sample_log.txt`, `sample_output.txt`, `sample_text.txt`, `sentiment_analysis.py`, `sentiment_prompts.py`, `sort_comparisons.py`, `text_file_creator.py`, `units_calculator.py`, `user_database.db`, `users.db`, and `word_frequency_count.py`.
- Status Bar:** Shows the current file path as `Ass-104.py` and other status indicators.

Prompt: 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:

- Code readability
- Naming conventions
- Formatting and style consistency
- Error handling
- Documentation quality
- 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:
 - Identified quality issues
 - Risk areas
 - Code smell detection
 - Improvement suggestions
- Optional improved version of the code
- Demonstration of AI as a code reviewer, not just a code generator

Note: Report should be submitted a word document for all tasks in a

Code: def d(a,b):return a/b

x=10

y=0

print(d(x,y))

AI Code Review Report

Readability

Issue: Cryptic function name d.

Risk: Hard for teams to understand.

Naming Conventions

Not following PEP 8.

Examples:

- a, b
- d