

Name:-U.Siddhartha
Hall No:-2403A54122

Task Description #1

(AI-Assisted Bug Detection)

Scenario:

A junior developer wrote the following Python function to calculate factorials:

```
def factorial(n):result = 1
```

```
for i in range(1, n):
```

result = result * ireturn result

- Run the code and test it with `factorial(5)`(expected output = 120).
 - Use AI (prompting) to review this code and identify the bug.
 - Ask AI to suggest corrections and rewrite the code.
 - Compare AI's corrected code with your own fix.

Input:-

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

- File Explorer:** Shows files like `EXPLORER`, `AI ALL LAB ASSIGNMENTS`, and several Python files including `10.4.py`.
- Editor Area:** The main pane displays the content of `10.4.py`. The code defines a function `factorial_buggy(n)` which calculates the factorial of n but misses the multiplication by n . It also includes a root cause analysis section.
- Terminal:** The terminal tab shows a command to run the file: `python 10.4.py`.
- Output:** The output tab shows the execution results of the command.
- Debug Console:** The debug console tab shows the execution results of the command.
- PowerShell:** A PowerShell window is open in the bottom right corner.
- Bottom Bar:** Includes tabs for `PROBLEMS`, `OUTPUT`, `DEBUG CONSOLE`, `TUTORIALS`, and `POR`. The `TUTORIALS` tab is active.
- Bottom Status:** Shows the current file is `10.4.py`, line 26, column 1, with 4 spaces, and UTF-8 encoding.
- Bottom Icons:** Includes icons for file operations, search, and various tools.

Output:-

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

- File Explorer:** Shows files like 5.2.py, task2.py, 10.4.py (selected), job_scoring.py, fibonacci_prompts.txt, fibonacci_recursive.py, results.json, secure_login.py, test_bias.py.
- Terminal:** Shows the command "powershell".
- Chat Panel:** Displays a conversation between AI and user. The user asks for help with a buggy code snippet. The AI provides a fixed version: "Fixed: $1 \times 1 \times 2 \times 3 \times 4 \times 5 = 120$ (CORRECT)".
- Code Editor:** The file 10.4.py is open, showing a buggy code snippet and its corrected version. The buggy code uses a range loop from 1 to n, while the corrected code uses a range loop from n to 1.
- Problems View:** Lists various issues found in the code, such as range loops and factorial calculations.
- Output View:** Shows build logs for file 10.4.py.
- Status Bar:** Shows the current file is 10.4.py, the environment is 3.13.7 (.venv), and the date is 28-11-2025.

Task Description #2 (Improving Readability & Documentation)

Scenario:

The following code works but is poorly written:

```
def calc(a,b,c):
```

```
    if c=="add":
```

```
        return a+b
```

```
    elif c=="sub":
```

```
        return a-b
```

```
    elif c=="mul":
```

```
        return a*b
```

```
    elif c=="div":
```

```
        return a/b
```

• Use AI to review this code

For readability, naming, and documentation issues.

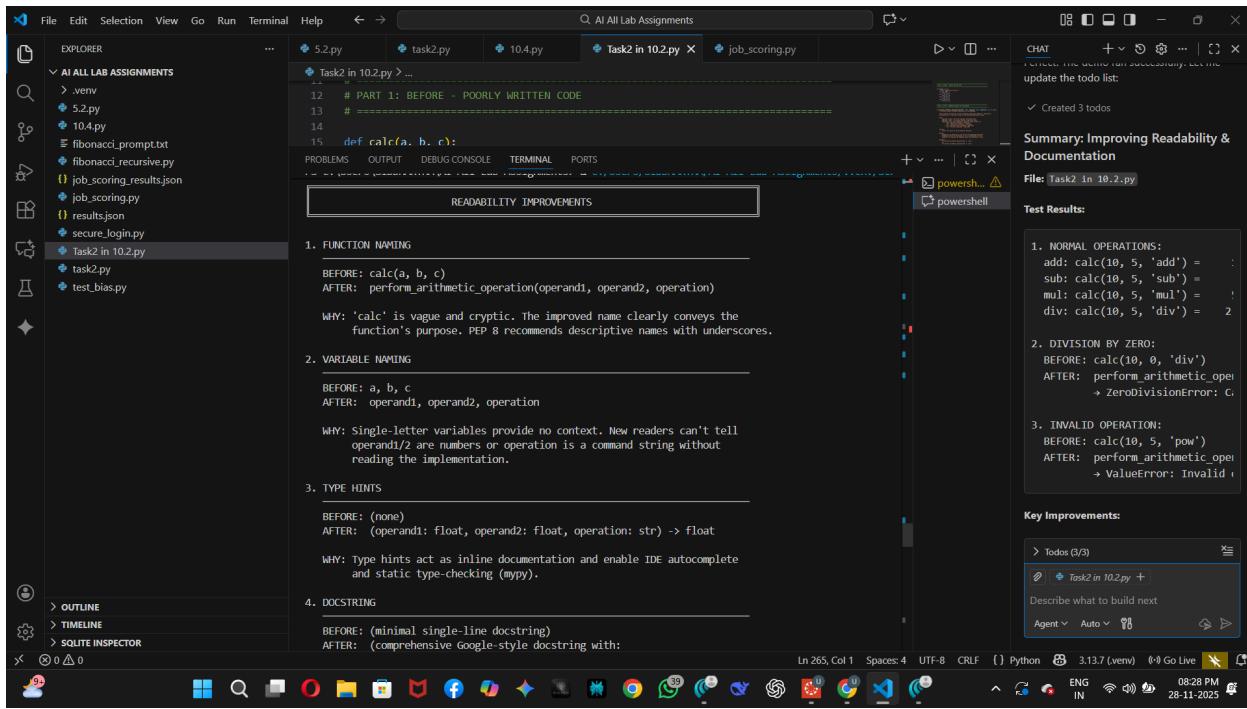
- Prompt AI to rewrite the code with:
 - Clear function & variable names.
 - Proper docstrings.
 - Exception handling for division by zero.
- Compare the before-and-after versions to evaluate AI's contribution.

Input:-

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows files like .venv, 5.2.py, 10.4.py, Task2 in 10.2.py, job_scoring.py, results.json, secure_login.py, task2.py, and test_bias.py.
- Code Editor:** Displays two versions of a Python file, Task2 in 10.2.py. The first version is titled "# PART 1: BEFORE - POORLY WRITTEN CODE" and contains a poorly written calc function. The second version is titled "# PART 2: AFTER - IMPROVED VERSION (AI REFACTORED)" and contains a well-documented perform_arithmetic_operation function.
- Output Panel:** Shows "CHAT" and "Test Results".
 - CHAT:** A log of AI interactions, including "Created 3 todos" and "Summary: Improving Readability & Documentation".
 - Test Results:** A table comparing normal operations (add, sub, mul, div), division by zero (calc(10, 0, 'div')), and invalid operations (calc(10, 5, 'pow')). It shows improvements from BEFORE to AFTER for each category.
- Bottom Status Bar:** Shows file path (Task2 in 10.2.py), line 265, column 1, spaces 4, encoding UTF-8, Python, and date/time (28-11-2025, 08:27 PM).

Output:-



Task Description #3 (Enforcing Coding Standards)

Scenario:

A team project requires following PEP8 style guide. Onedeveloper submits:

```
def Checkprime(n):
    for i in range(2,n):
        if n%i==0:
            return False
    return True
```

- Run this code and verify correctness.
- Use AI to perform a code quality review for PEP8 compliance.
- Prompt AI to return a refactored version with proper indentation, spacing, and naming conventions.
- Discuss how automated AI review can save time in large-scale projects

Input:-

AI All Lab Assignments

Task3 in 10.2.py > ...

```

13 # PART 1: ORIGINAL CODE (PEP8 VIOLATIONS)
14 # =====
15
16 def Checkprime(n):
17     """Original function with PEP8 violations."""
18     for i in range(2,n):
19         if n % i == 0:
20             return False
21     return True
22
23
24 # =====
25 # PART 2: PEP8 VIOLATIONS IDENTIFIED
26 # =====
27
28 PEP8_VIOLATIONS = """
29
30     PEP8 VIOLATIONS DETECTED
31
32
33 1. FUNCTION NAMING (PEP8 E225, E501)
34
35 Violation: def Checkprime(n):
36 Rule: Function names should be lowercase with underscores (snake_case)
37 PEP8: "Function names should be lowercase, with words separated by
38 underscores as necessary to improve readability."
39 Fix: def check_prime(n):
40
41 2. WHITESPACE AROUND OPERATORS (PEP8 E225)
42
43 Violation: n % i

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Test Results: Original=0 failures, Refactored=0 failures
PS C:\Users\SIDDHARTH\AI All Lab Assignments> []

Ln 379, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.7 (venv) 08:32 PM 28-11-2025 ENG IN

Output:-

AI All Lab Assignments

Task3 in 10.2.py > ...

```

13 # PART 1: ORIGINAL CODE (PEP8 VIOLATIONS)
14 # =====
15
16 def Checkprime(n):
17     """Original function with PEP8 violations."""
18     for i in range(2,n):
19         if n % i == 0:
20             return False
21     return True
22
23
24 # =====
25 # PART 2: PEP8 VIOLATIONS IDENTIFIED
26 # =====
27
28 PEP8_VIOLATIONS = """
29
30     PEP8 VIOLATIONS DETECTED
31
32
33 1. FUNCTION NAMING (PEP8 E225, E501)
34
35 Violation: def Checkprime(n):
36 Rule: Function names should be lowercase with underscores (snake_case)
37 PEP8: "Function names should be lowercase, with words separated by
38 underscores as necessary to improve readability."
39 Fix: def check_prime(n):
40
41 2. WHITESPACE AROUND OPERATORS (PEP8 E225)
42
43 Violation: n % i

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SIDDHARTH\AI All Lab Assignments> "C:\Users\SIDDHARTH\AI All Lab Assignments\venv\Scripts\python.exe" "C:\Users\SIDDHARTH\AI All Lab Assignments\Task3 in 10.2.py"

Criteria	Original	Refactored	Notes
Function naming (snake_case)	FAIL	PASS	Checkprime + check_prime
Spacing around operators	FAIL	PASS	n % i == 0 + n % i == 0
Spacing after commas	FAIL	PASS	range(2,n) + range(2, n)
Comprehensive docstring	FAIL	PASS	Added Args, Returns, Raises, Examples
Type hints	FAIL	PASS	Added type annotations
Input validation	FAIL	PASS	Added error handling
Edge cases handled	FAIL	PASS	n < 2 properly handled
Line length	PASS	PASS	All lines readable
Code efficiency	FAIL	PASS	Optimized to O(sqrt(n))
Readability	PASS	PASS	Improved clarity
Comprehensive docstring	FAIL	PASS	Added Args, Returns, Raises, Examples
Type hints	FAIL	PASS	Added type annotations
Input validation	FAIL	PASS	Added error handling
Edge cases handled	FAIL	PASS	n < 2 properly handled
Line length	PASS	PASS	All lines readable
Code efficiency	FAIL	PASS	Optimized to O(sqrt(n))
Readability	PASS	PASS	Improved clarity
Edge cases handled	FAIL	PASS	n < 2 properly handled
Line length	PASS	PASS	All lines readable
Code efficiency	FAIL	PASS	Optimized to O(sqrt(n))
Readability	PASS	PASS	Improved clarity
Readability	PASS	PASS	Optimized to O(sqrt(n))
Readability	PASS	PASS	Improved clarity

AI Code Review Benefits for Large Teams:

- TIME SAVINGS:** 15-30 min manual review per PR → <1 sec AI pre-check
- CONSISTENCY:** All code follows same style across team
- FOCUS:** Humans review logic/architecture, AI handles style
- SCALABILITY:** Works for 10 or 1000 developers equally well
- LEARNING:** Junior devs get instant feedback on best practices
- QUALITY:** Catches bugs early (like the n=1 edge case here)
- CI/CD:** Automated checks prevent bad code from shipping
- METRICS:** Track code quality trends and improvement areas

All tasks complete! The comprehensive analysis shows how AI-driven code review combined with automated tools can dramatically improve code quality, consistency, and team productivity.

Task3 in 10.2.py + Describe what to build next Agent Auto 3.13.7 (venv) 08:33 PM 28-11-2025 ENG IN

Task Description #4

(AI as a Code Reviewer in Real Projects)

Scenario:

You are part of a GitHub project. A teammate submits this pull request:

```
def processData(d):
    return [x*2 for x in d if x%2==0]
```

- Review this function manually for readability, reusability, and edge cases.

- Use AI to generate a code review comment, focusing on:

- Naming conventions.

- Input validation (e.g., what if d is not a list?).

- Adding type hints.

- Modify the function based on AI's suggestions.

- Write a short reflection: Would you trust AI as a stand-alone reviewer, or only as a support to

Input:-

```
task4 in 10.2.py > process_even_numbers
1 from typing import List, Iterable
2
3 def process_even_numbers(data: Iterable[int]) -> List[int]:
4     """
5         Processes the input iterable, returning a list of doubled values for even integers.
6
7     Args:
8         data (Iterable[int]): An iterable of integers.
9
10    Returns:
11        List[int]: A list containing doubled values of even integers from the input.
12    """
13    if not isinstance(data, Iterable):
14        raise TypeError("Input must be an iterable of integers.")
15    result = []
16    for x in data:
17        if not isinstance(x, int):
18            raise ValueError("All elements must be integers.")
19        if x % 2 == 0:
20            result.append(x * 2)
21
22    return result
```

The error occurred because the function `process_even_numbers` is not defined in the current execution context. To run the function, you need to include its definition along with the test code in the same snippet.

Would you like me to run the function with a sample input and show the output? If yes, I will execute both the function definition and a test print statement together.

Output:-