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

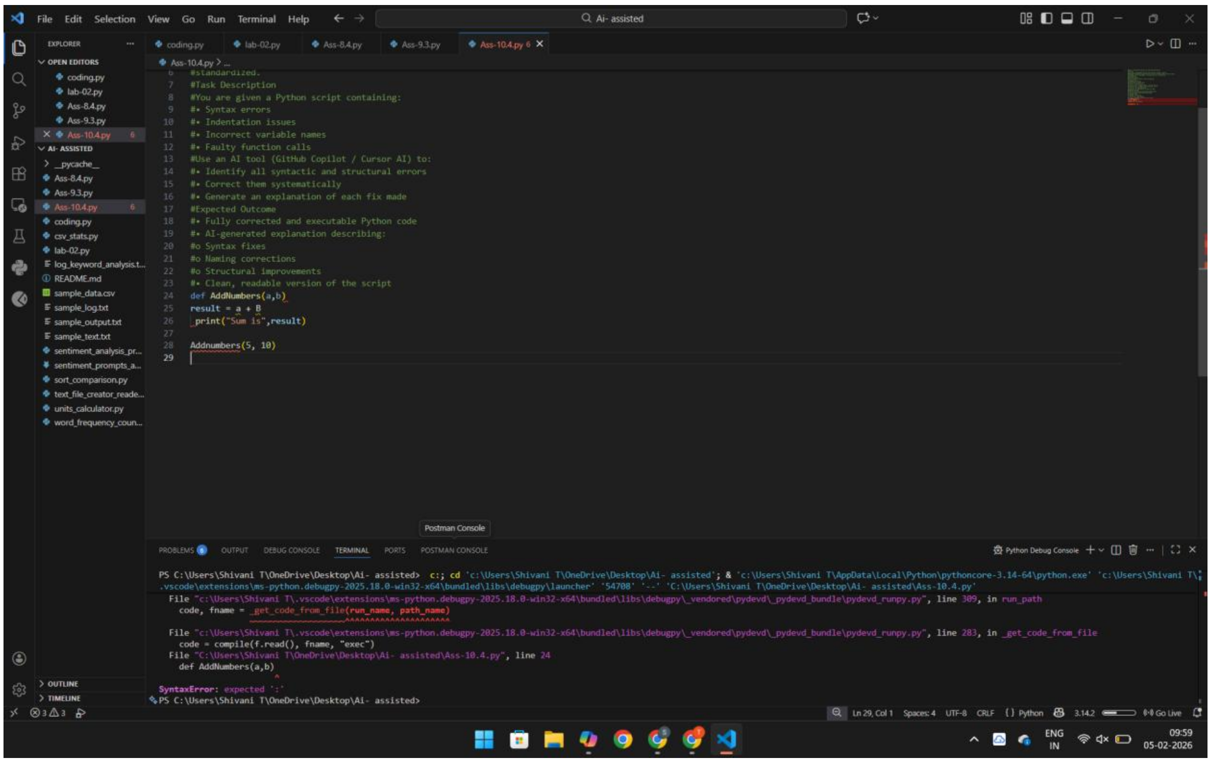
# Structural improvements

# Clean, readable version of the script

Code:

Error code

```
def AddNumbers(a,b)

result = a + B

 print("Sum is",result)

Addnumbers(5, 10)
```

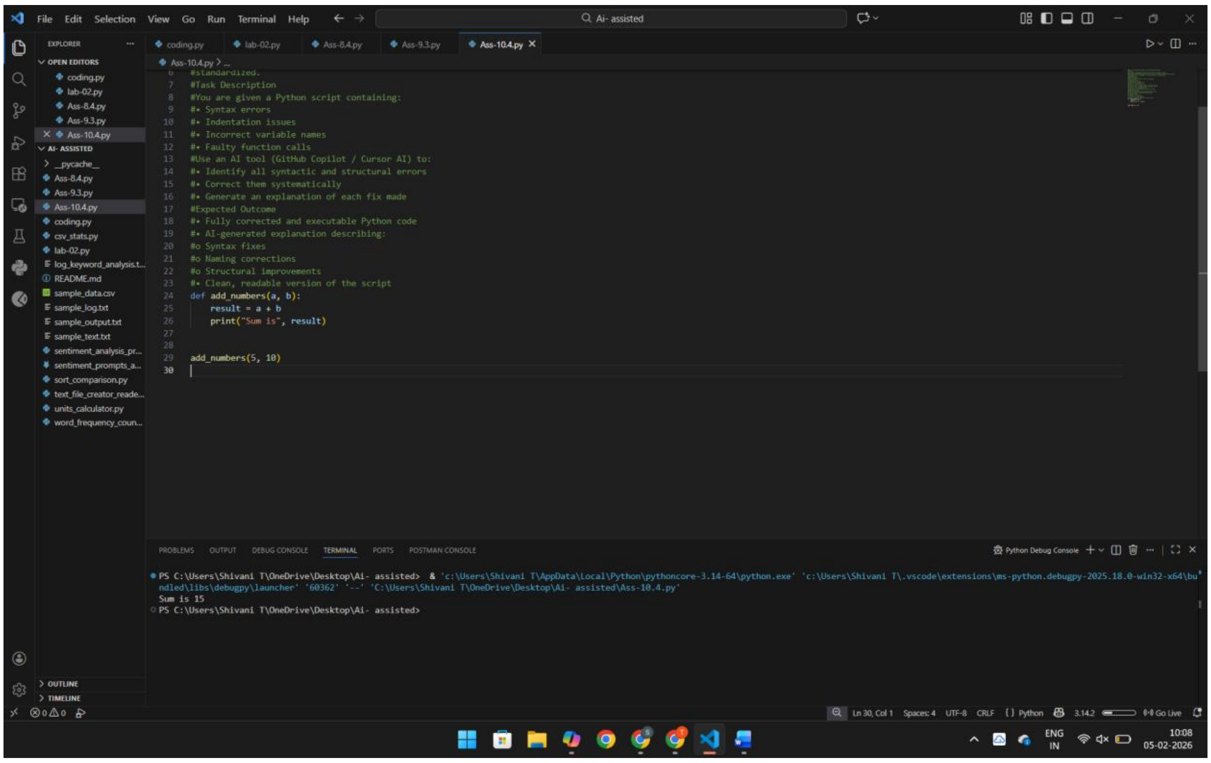


```
#standardized.
#Task Description
#You are given a Python script containing:
#• Syntax errors
#• Indentation issues
#• Incorrect variable names
#• Faulty function calls
#Use an AI tool (GitHub Copilot / Cursor AI) to:
#• Identify all syntactic and structural errors
#• Correct them systematically
#• Generate an explanation of each fix made
#Expected Outcome
#• Fully corrected and executable Python code
#• AI-generated explanation describing:
#o Syntax fixes
#o Naming corrections
#o Structural improvements
#• Clean, readable version of the script
def AddNumbers(a,b)
result = a + B
 print("Sum is",result)

Addnumbers(5, 10)
```

```
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>  c:; cd 'c:\Users\Shivani T\OneDrive\Desktop\Ai- assisted'; & 'c:\Users\Shivani T\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54708' '--' 'C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py'
  File "c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\_vendored\pydevd\_pydevd_bundle\pydevd_runpy.py", line 309, in run_path
    code, fname = _get_code_from_file(run_name, path_name)
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\_vendored\pydevd\_pydevd_bundle\pydevd_runpy.py", line 283, in _get_code_from_file
    code = compile(f.read(), fname, "exec")
  File "C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py", line 24
    def AddNumbers(a,b)
                       ^
SyntaxError: expected ':'
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>
```

Correct code:

```
def add_numbers(a, b):
    result = a + b
    print("Sum is", result)


add_numbers(5, 10)
```



```
#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
#o Naming corrections
#o Structural improvements
#• Clean, readable version of the script
def add_numbers(a, b):
    result = a + b
    print("Sum is", result)


add_numbers(5, 10)
```

```
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>  & 'c:\Users\Shivani T\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60362' '--' 'C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py'
Sum is 15
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>
```

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

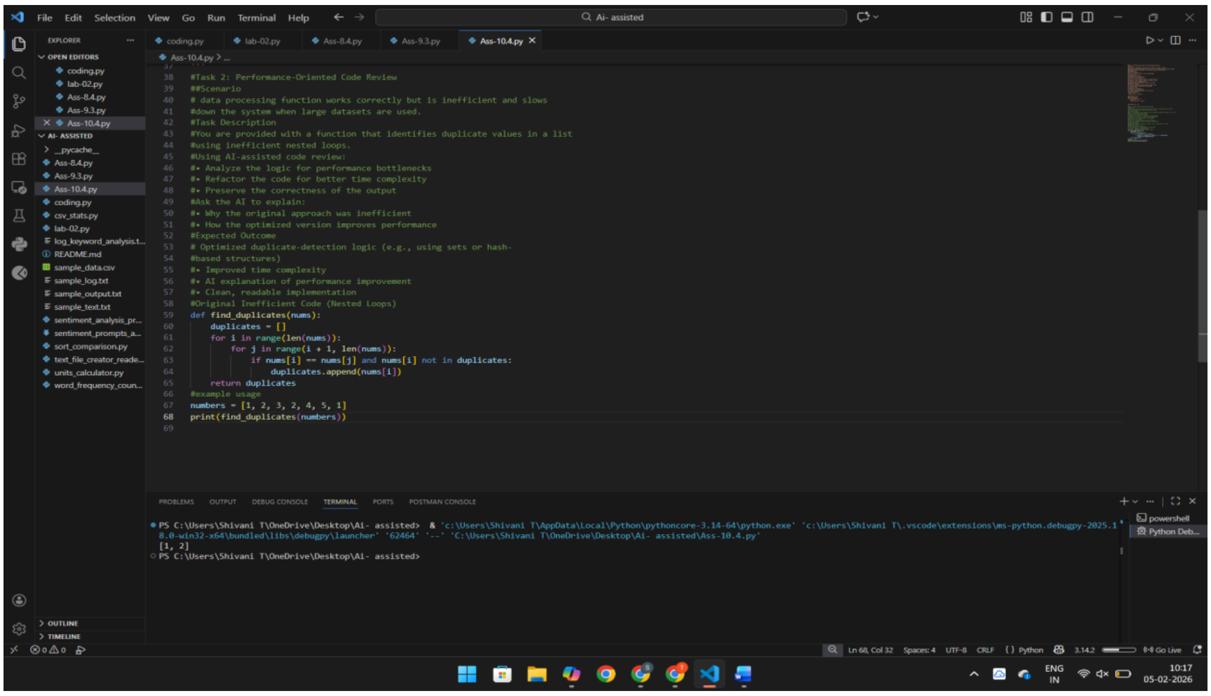
#• AI explanation of performance improvement

#• 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))v
```

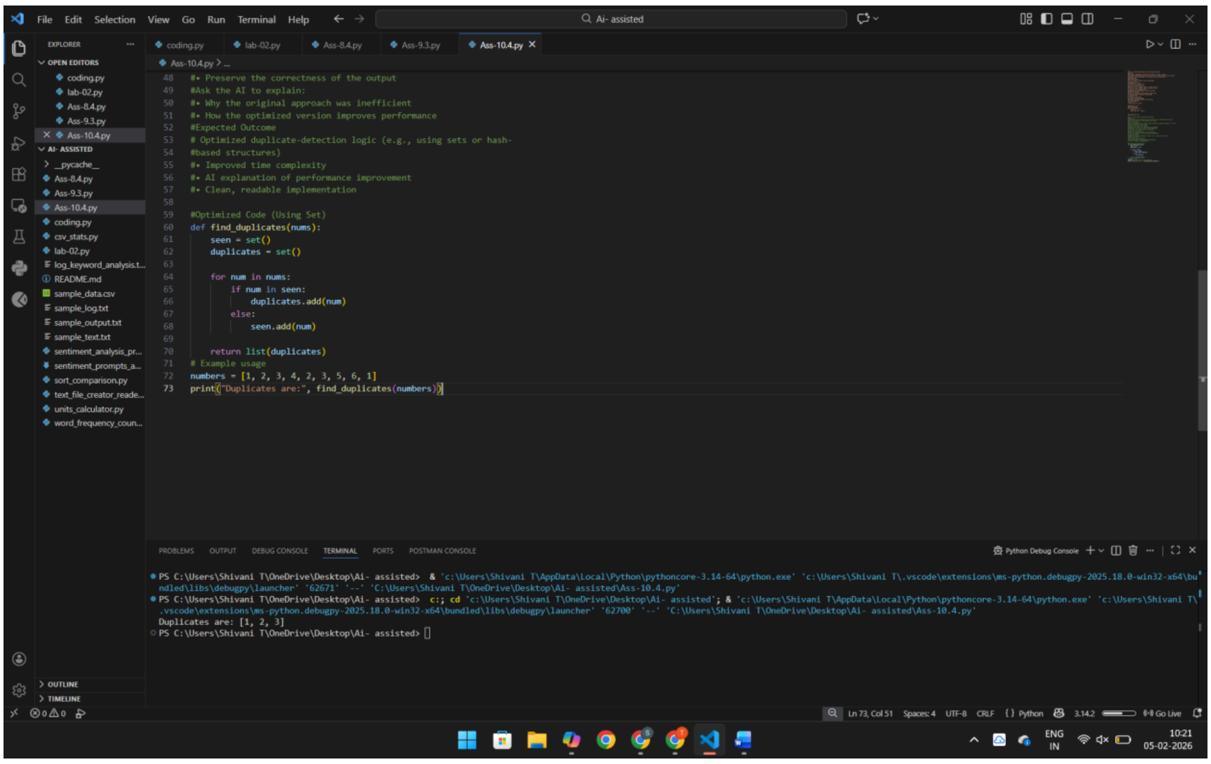


```
#Task 2: Performance-Oriented Code Review
##Scenario
# data processing function works correctly but is inefficient and slows
#down the system when large datasets are used.
#Task Description
#You are provided with a function that identifies duplicate values in a list
#using inefficient nested loops.
#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
#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))
```

```
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>  & 'c:\Users\Shivani T\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62464' '--' 'C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py'
[1, 2]
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>
```

Optimized Code (Using Set)

```python
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))
```



```python
#• 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

#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))
```

```
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>  & 'c:\Users\Shivani T\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62671' '--' 'C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py'
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>  c:; cd 'c:\Users\Shivani T\OneDrive\Desktop\Ai- assisted'; & 'c:\Users\Shivani T\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62700' '--' 'C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py'
Duplicates are: [1, 2, 3]
PS C:\Users\Shivani T\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 to3poor 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:
```

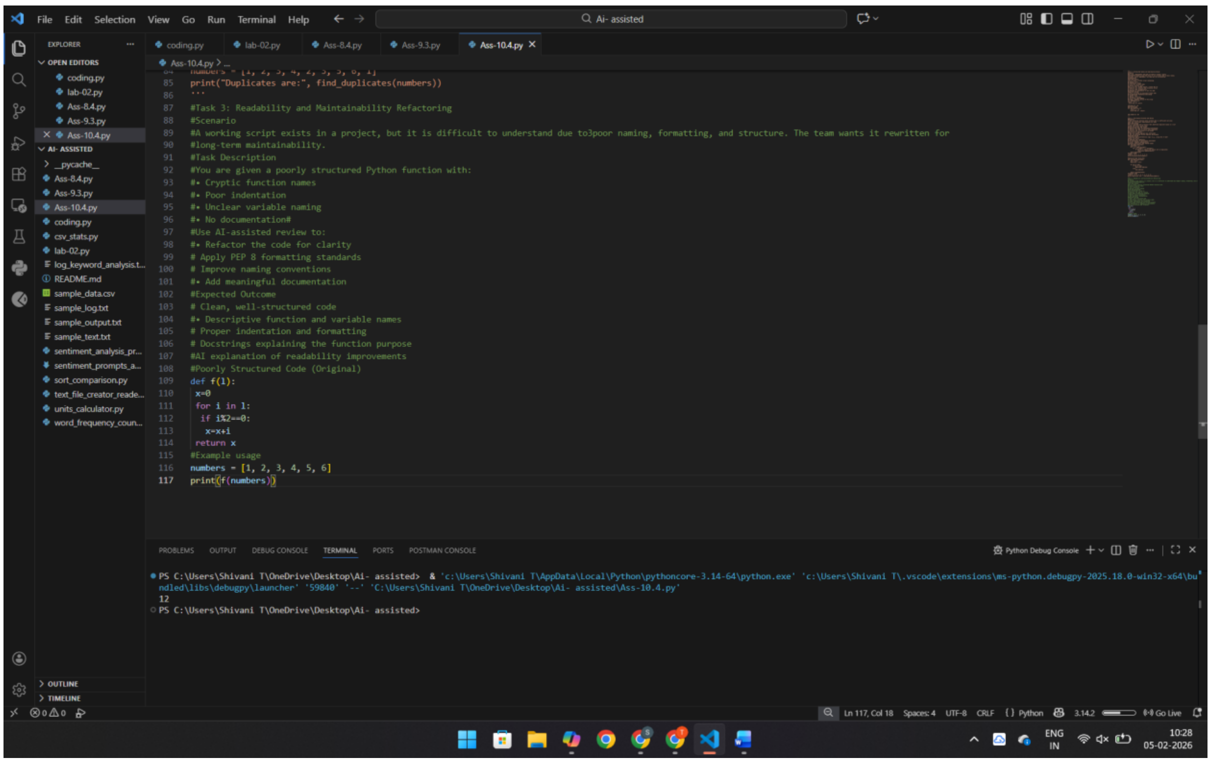
   x=x+i

  return x

#Example usage

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

print(f(numbers))



```
print("Duplicates are:", find_duplicates(numbers))
'''
#Task 3: Readability and Maintainability Refactoring
#Scenario
#A working script exists in a project, but it is difficult to understand due to3poor 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
#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))
```

```
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>  & 'c:\Users\Shivani T\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59840' '--' 'C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py'
12
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>
```

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:

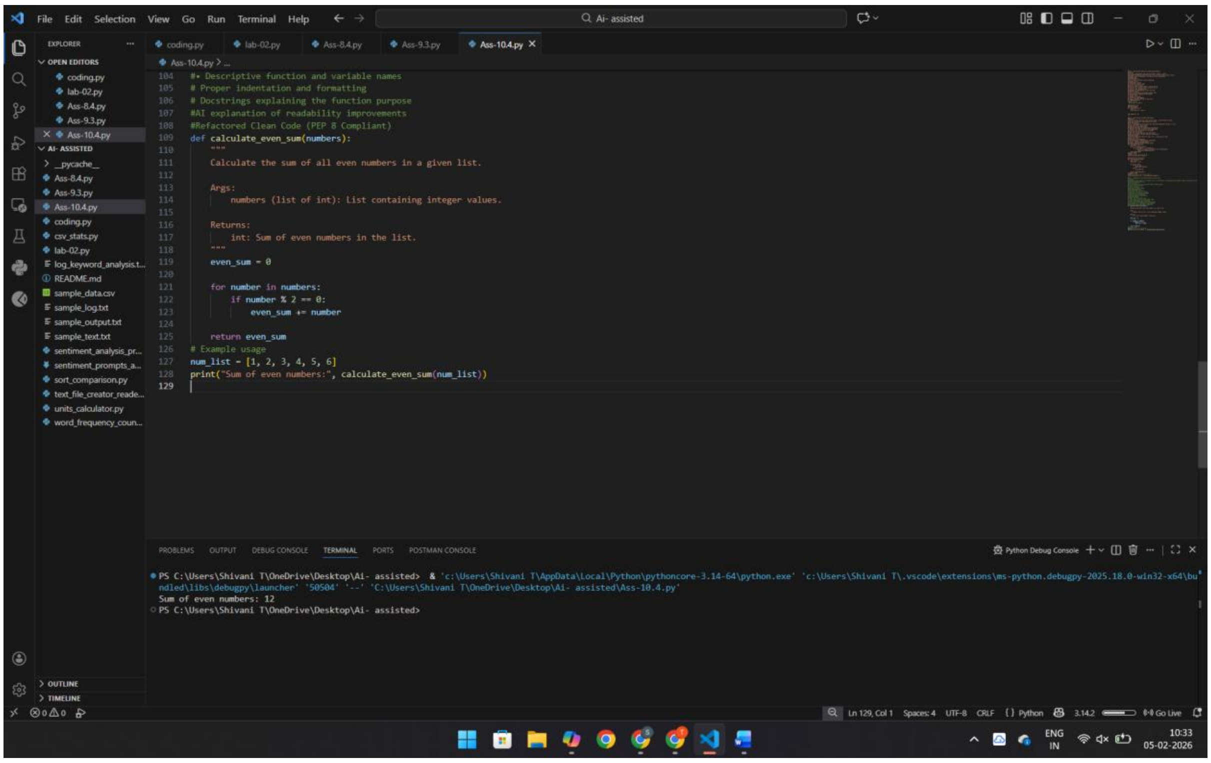
    int: Sum of even numbers in the list.

  """

  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))
```



```
#• Descriptive function and variable names
# Proper indentation and formatting
# Docstrings explaining the function purpose
#AI explanation of readability improvements
#Refactored Clean Code (PEP 8 Compliant)
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))
```

```
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>  & 'c:\Users\Shivani T\AppData\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\Shivani T\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50504' '--' 'C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted\Ass-10.4.py'
Sum of even numbers: 12
PS C:\Users\Shivani T\OneDrive\Desktop\Ai- assisted>
```

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

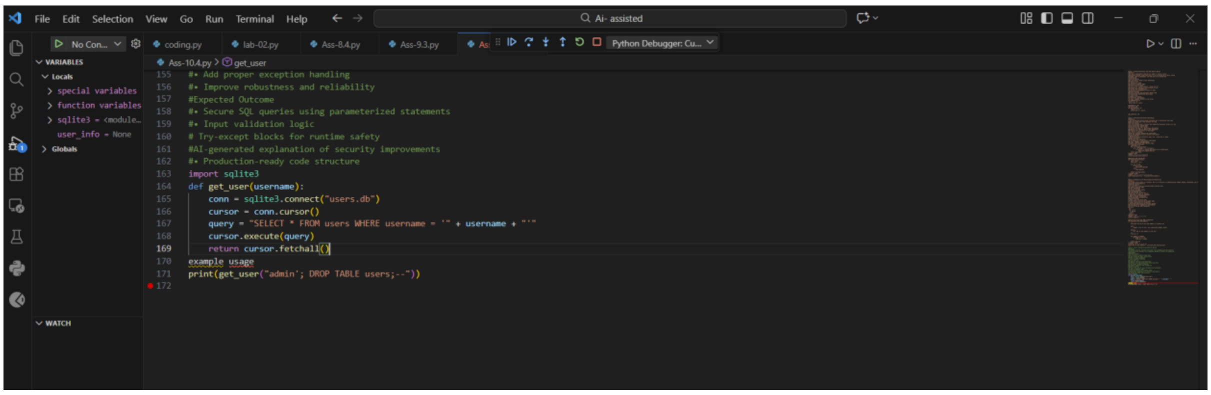
Try-except blocks for runtime safety

#AI-generated explanation of security improvements

#• Production-ready code structure

Code:

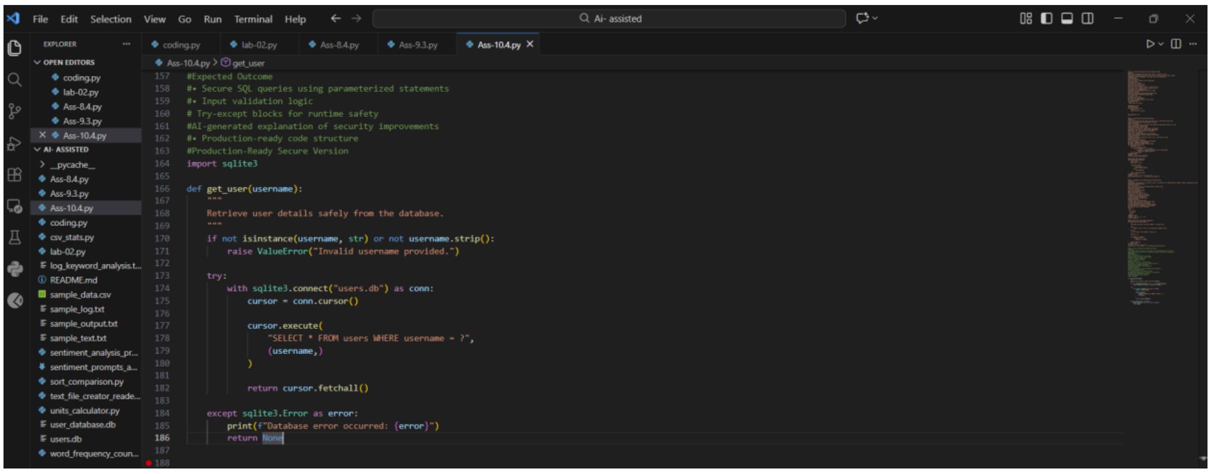
```
import sqlite3

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()
```



```
#• 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
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()
example usage
print(get_user("admin'; DROP TABLE users;--"))
```

```
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
```


File Edit Selection View Go Run Terminal Help
Ai- assisted
coding.py lab-02.py Ass-8.4.py Ass-9.3.py Ass-10.4.py
Ass-10.4.py > get_user

```
#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
#Production-Ready Secure Version
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
```

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:

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

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