SR UNIVERSITY

AI ASSISTED CODING

NAME:BEGALA HASINI
2503A51L13

BATCH:19

Using AI to Improve Code Quality and Readability

Lab Objectives

- Use AI for automated code review and quality enhancement.
- Identify and fix syntax, logical, performance, and security issues in Python code.
- Improve readability and maintainability through structured refactoring and comments.
- Apply prompt engineering for targeted improvements.
- Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices

TASK 1: Syntax and Error Detection

Identify and fix syntax, indentation, and variable errors in the given script.

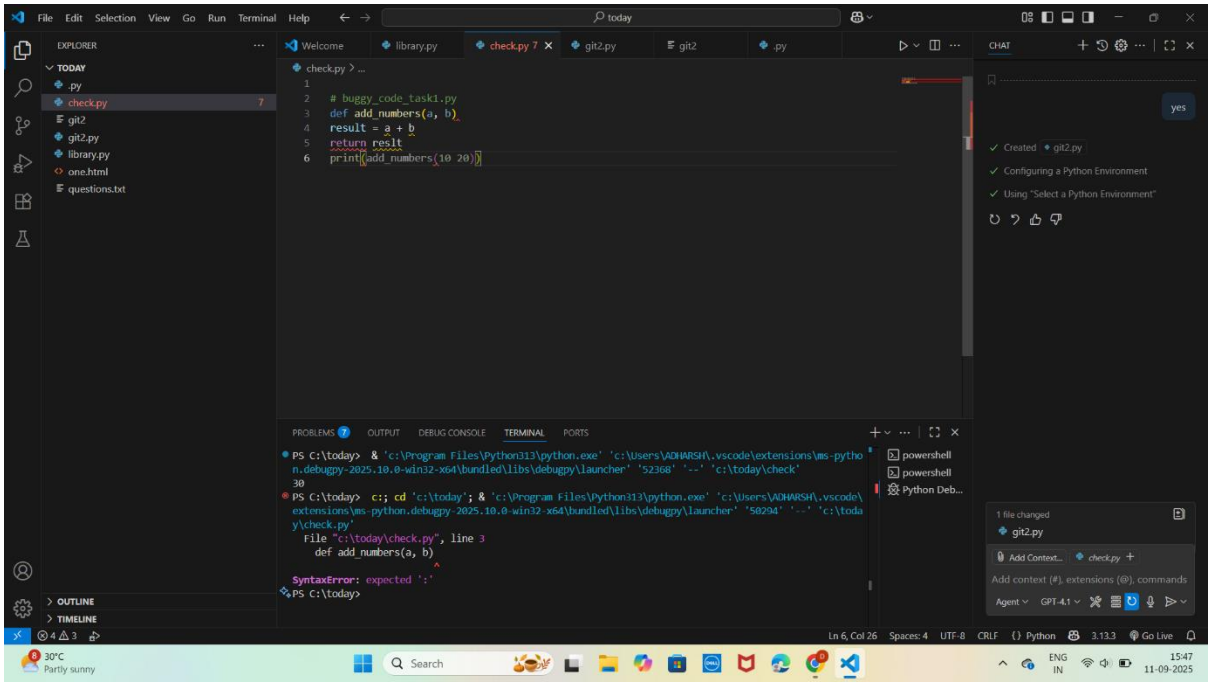
```
# buggy_code_task1.py
def add_numbers(a, b)
    result = a + b
    return reslt
print(add_numbers(10 20))
```

Expected Output:

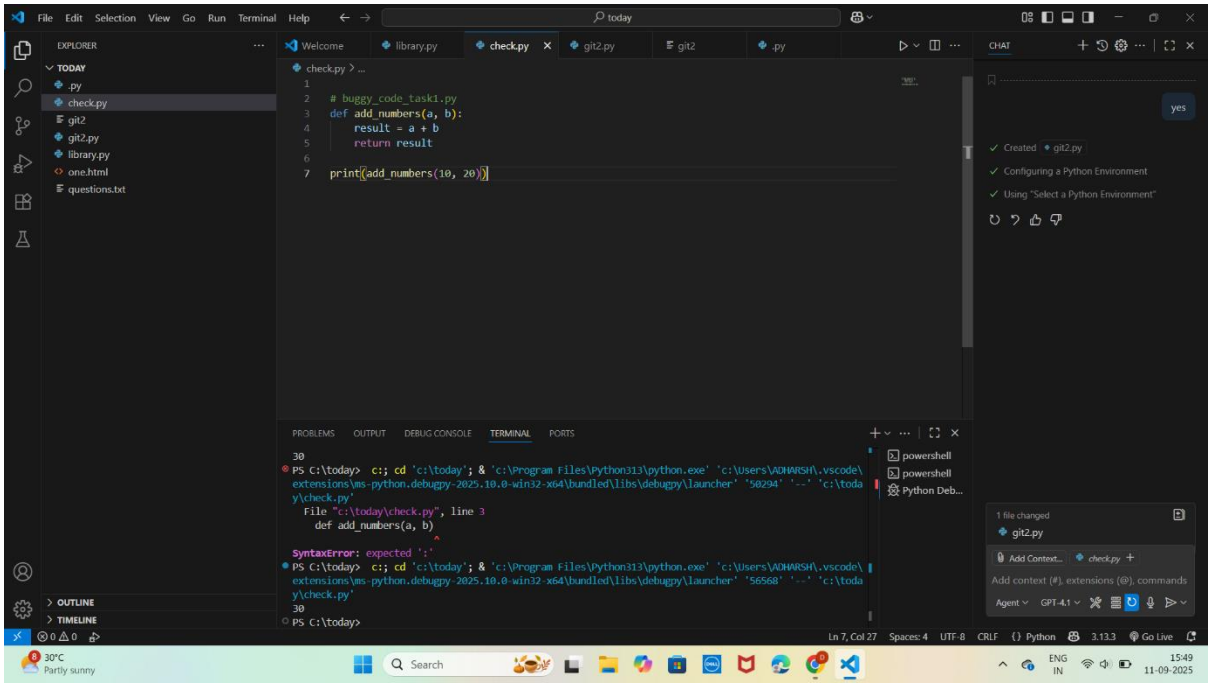
- Corrected code with proper syntax (: after function, fixed variable name, corrected function call).

- AI should explain what was fixed.

Before Detection:



After Detection:



Task 2: Logical and Performance Issue Review

Optimize inefficient logic while keeping the result correct.

buggy_code_task2.py

```
def find_duplicates(nums):
    duplicates = []
    for i in range(len(nums)):
        for j in range(len(nums)):
            if i != j and nums[i] == nums[j] and nums[i] not in duplicates:
                duplicates.append(nums[i])
    return duplicates
numbers = [1,2,3,2,4,5,1,6,1,2]
print(find_duplicates(numbers))
```

Expected Output:

More efficient duplicate detection (e.g., using sets).

AI should explain the optimization.

Before detection:

```
task2.py > ...

# buggy_code_task2.py
def find_duplicates(nums):
duplicates = []
for i in range(len(nums)):
for j in range(len(nums)):
if i != j and nums[i] == nums[j] and nums[i] not in duplicates:
duplicates.append(nums[i])
return duplicates
numbers = [1,2,3,2,4,5,1,6,1,2]
print(find_duplicates(numbers))
```

```
extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64351' '--' 'c:\toda
y\task2.py'
  File "c:\today\task2.py", line 4
    duplicates = []
    ^^^^^^^^^^
IndentationError: expected an indented block after function definition on line 3
```

After Detection:

```
task2.py > ...
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)

numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
print(find_duplicates(numbers))
```

```
extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '63701' '--' 'c:\toda
y\task2.py'
[1, 2]
PS C:\today>
```

Task 3: Code Refactoring for Readability

Refactor messy code into clean, PEP 8–compliant, well-structured code.

```
# buggy_code_task3.py
def c(n):
 x=1
 for i in range(1,n+1):
  x=x*i
 return x
print(c(5))
```

Expected Output:

Function renamed to calculate_factorial.

Proper indentation, variable naming, docstrings, and formatting.

AI should provide a more readable version.

Before Detection:

```
task3.py > ...
# buggy_code_task3.py
def c(n):
x=1
for i in range(1,n+1):
x=x*i
return x
print(c(5))
```

```
extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '59087' '--' 'c:\today\task3.py'
  File "c:\today\task3.py", line 3
    x=1
    ^
IndentationError: expected an indented block after function definition on line 2
```

After Detection:

```
task3.py > ...
# buggy_code_task3.py
def c(n):
    x = 1
    for i in range(1, n + 1):
        x = x * i
    return x

print(c(5))
```

```
extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64927' '--' 'c:\today\task3.py'
120
```

Task 4: Security and Error Handling Enhancement

Add security practices and exception handling to the code.

```
# buggy_code_task4.py

import sqlite3

def get_user_data(user_id):

    conn = sqlite3.connect("users.db")

    cursor = conn.cursor()
```

```
    query = f"SELECT * FROM users WHERE id = {user_id};"  # Potential SQL injection risk
    cursor.execute(query)
    result = cursor.fetchall()
    conn.close()
    return result
user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

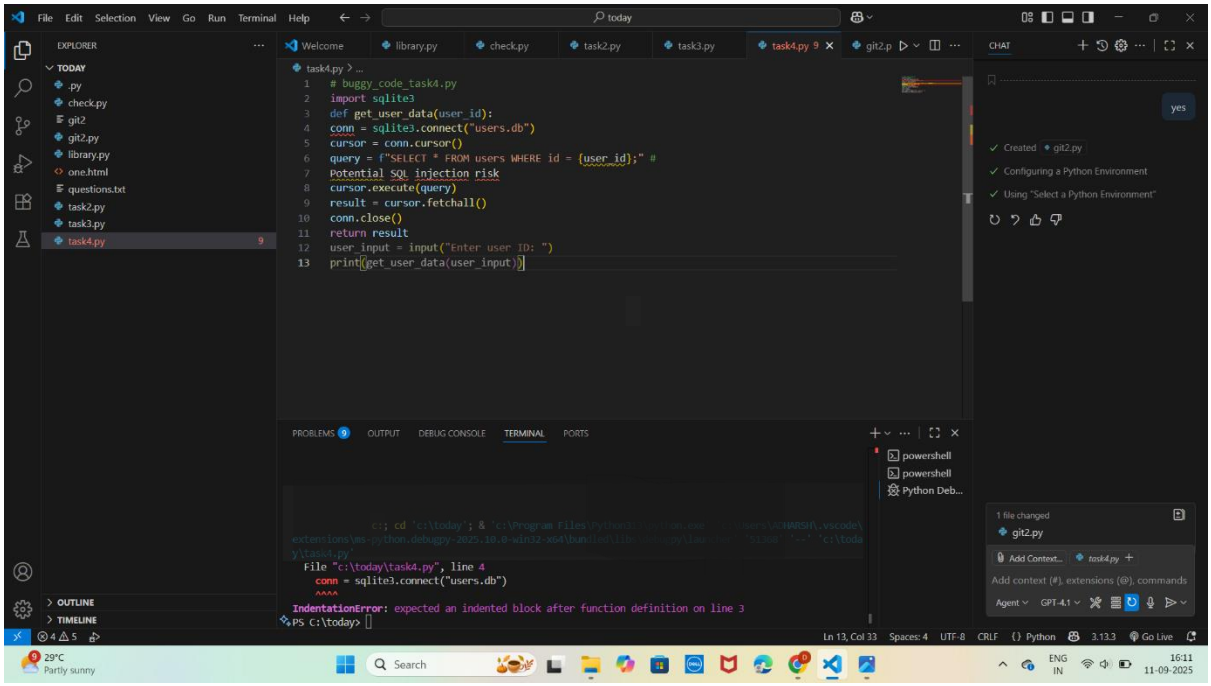
Expected Output:

Safe query using parameterized SQL (? placeholders).

Try-except block for database errors.

Input validation before query execution.

Before Detection:



After Detection:

```python
task4.py > ...
# buggy_code_task4.py
import sqlite3
def get_user_data(user_id):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE id = ?;"
    cursor.execute(query, (user_id,))
    result = cursor.fetchall()
    conn.close()
    return result

if __name__ == "__main__":
    user_input = input("Enter user ID: ")
    print(get_user_data(user_input))
```

```
n.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '56895' '--' 'c:\today\task4.py'
Enter user ID: 1234
```

Task 5: Automated Code Review Report Generation

Generate a review report for this messy code.

```
# buggy_code_task5.py
def calc(x,y,z):
 if z=="add":
  return x+y
 elif z=="sub": return x-y
 elif z=="mul":
  return x*y
 elif z=="div":
  return x/y
 else: print("wrong")

print(calc(10,5,"add"))
print(calc(10,0,"div"))
```

Expected Output:

AI-generated review report should mention:

Missing docstrings

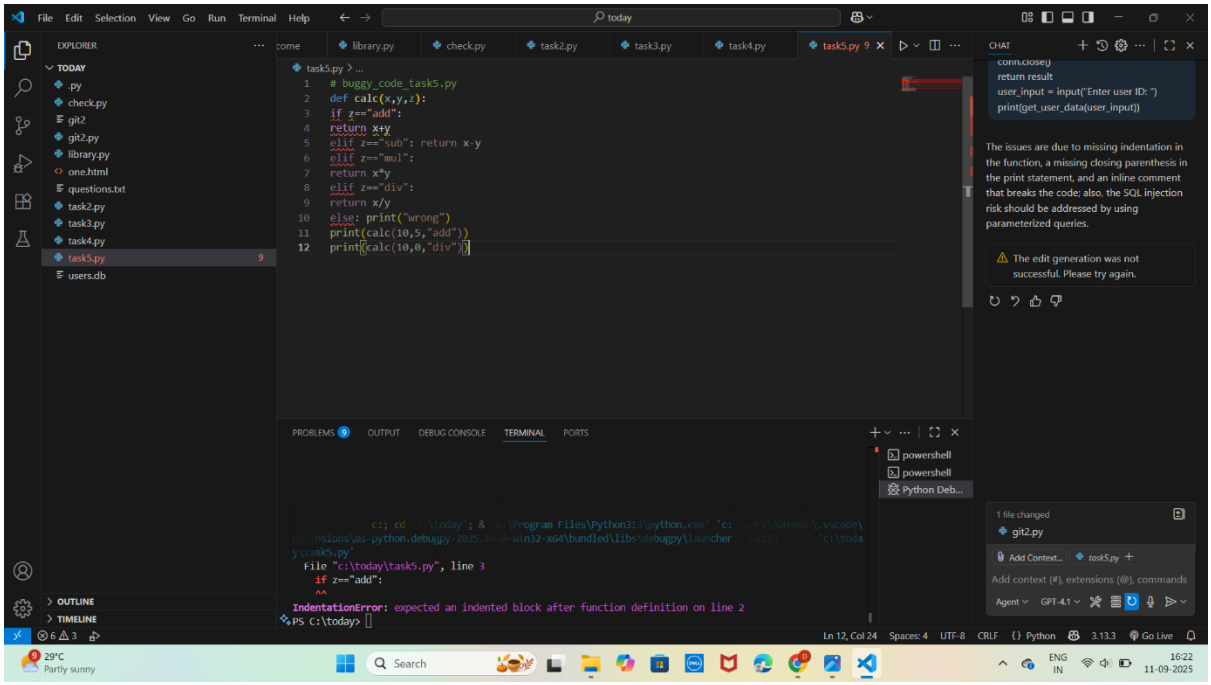
Inconsistent formatting (indentation, inline return)

Missing error handling for division by zero

Non-descriptive function/variable names

Suggestions for readability and PEP 8 compliance

Before Detection:



After Detection:

```
# Review Report for buggy_code_task5.py

## Issues Identified

1. **Indentation Errors**: The function body and control flow statements are not properly inde
2. **Inconsistent Return Statements**: The `else` block prints "wrong" but does not return a v
3. **Division by Zero**: The call `calc(10,0,"div")` will raise a `ZeroDivisionError`.
4. **Inconsistent Code Style**: Some `elif` statements are on the same line as their code, oth
5. **Lack of Input Validation**: No checks for input types or valid operation strings.
6. **No Docstring**: The function lacks a docstring explaining its purpose and parameters.

## Suggestions

- Fix indentation for all code blocks.
- Ensure all branches of the function return a value.
- Handle division by zero with a try-except block or input check.
- Use consistent code style for readability.
- Add a docstring to the function.
- Optionally, validate input types and supported operations.
```

task5.py > ...

```python
# buggy_code_task5.py
def calc(x, y, z):
    if z == "add":
        return x + y
    elif z == "sub":
        return x - y
    elif z == "mul":
        return x * y
    elif z == "div":
        if y != 0:
            return x / y
        else:
            return "Error: Division by zero"
    else:
        return "Error: Invalid operation"

print(calc(10, 5, "add"))
print(calc(10, 0, "div"))
```

```
extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64760' '--' 'c
y\task5.py'
15
```

```
extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64760' '--' 'c:\toda
y\task5.py'
15
Error: Division by zero
PS C:\today>
```