AI ASSISTED CODING

Lab Assignment-10.4

Name : T.Shylasri

H.T.NO: 2303A51876

Batch-14(LAB-10)

Date: 19-02-2026

ASSIGNMENT-10.4

Lab 9 – Code Review and Quality: 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: 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:

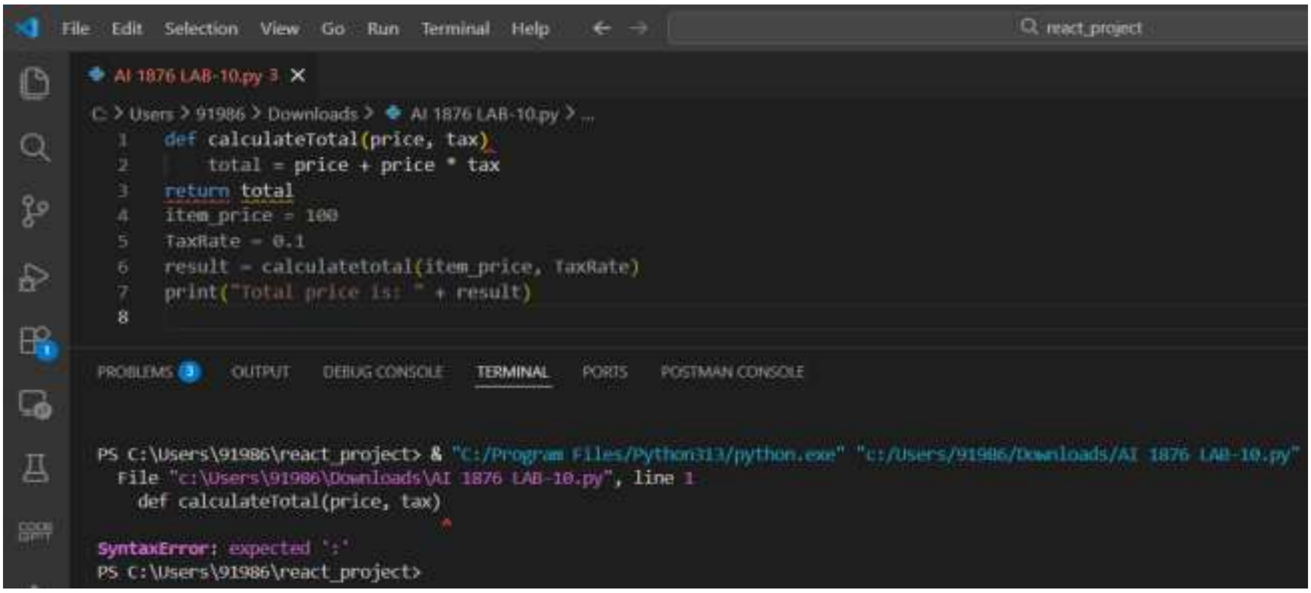
o Syntax fixes

o Naming corrections

o Structural improvements

- Clean, readable version of the script

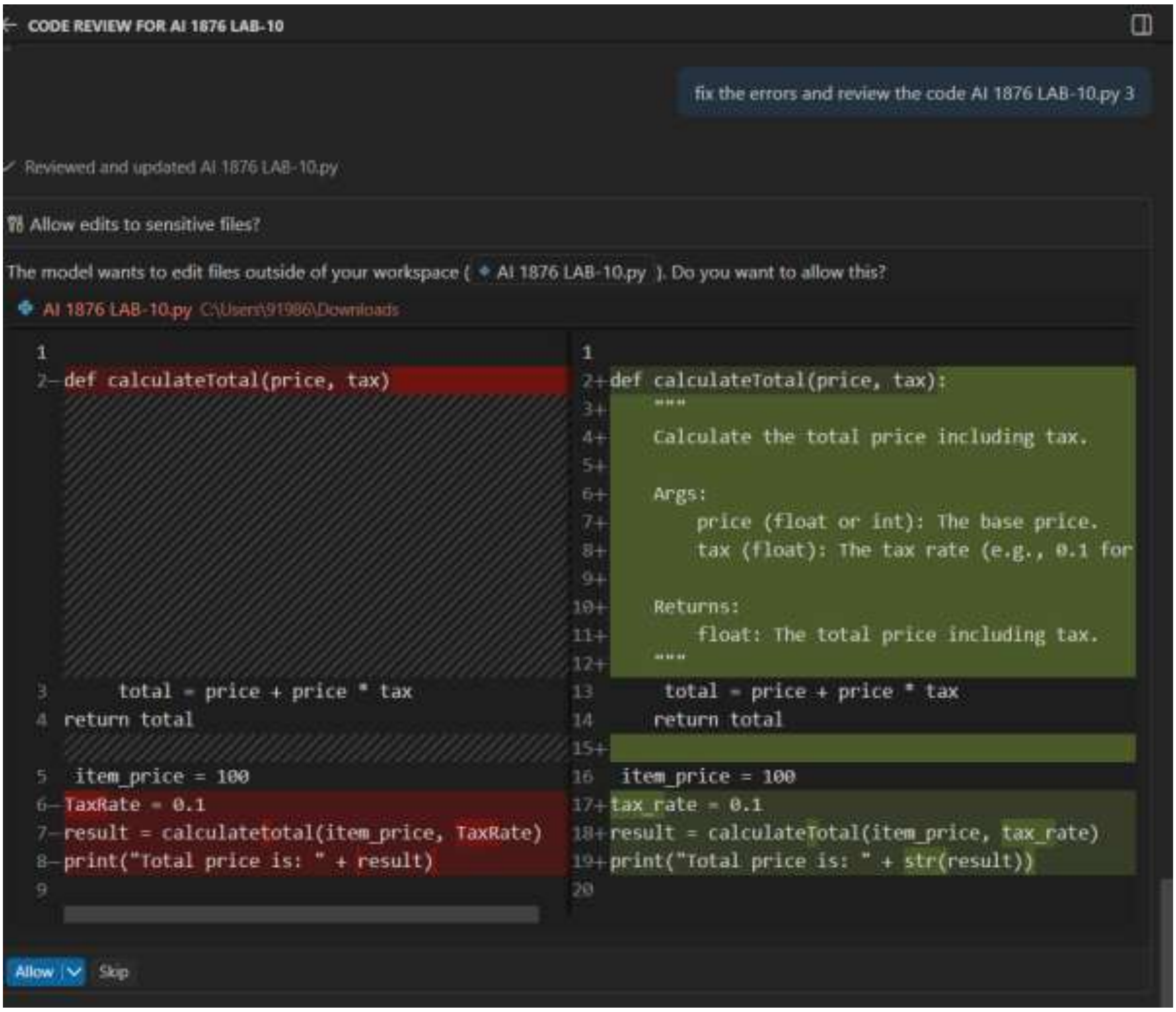
Error code screenshot:



```python
def calculateTotal(price, tax)
    total = price + price * tax
return total
item_price = 100
TaxRate = 0.1
result = calculatetotal(item_price, TaxRate)
print("Total price is: " + result)
```

```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
  File "c:\Users\91986\Downloads\AI 1876 LAB-10.py", line 1
    def calculateTotal(price, tax)
                                  ^
SyntaxError: expected ':'
PS C:\Users\91986\react_project>
```

Review code:



fix the errors and review the code AI 1876 LAB-10.py 3

✓ Reviewed and updated AI 1876 LAB-10.py

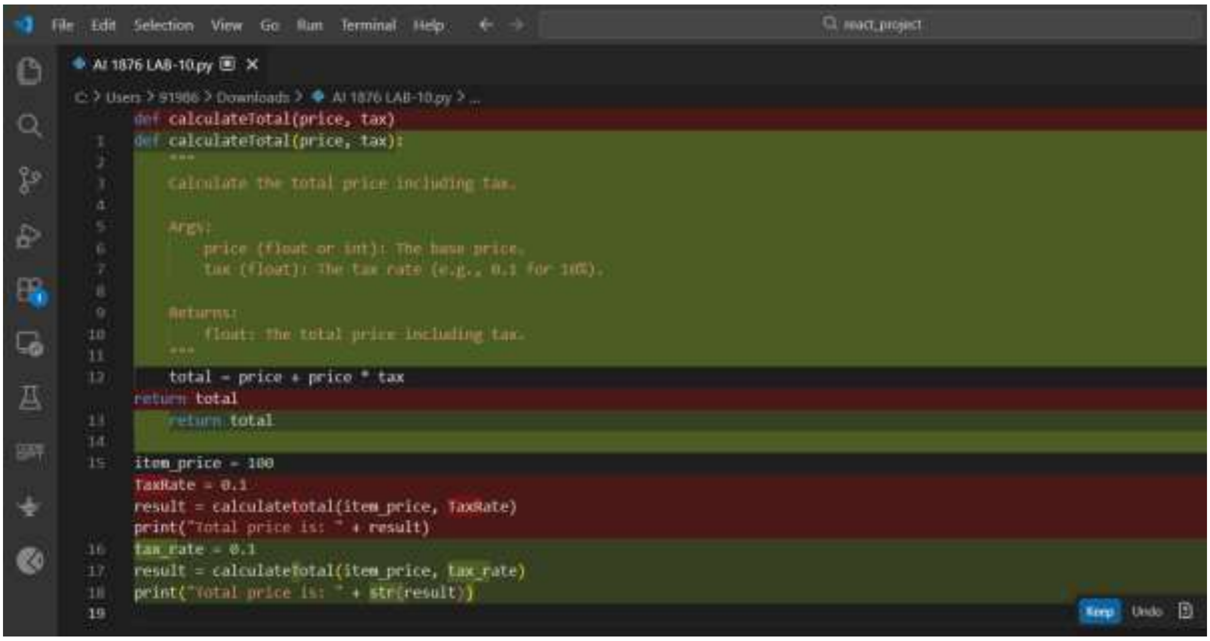
Allow edits to sensitive files?

The model wants to edit files outside of your workspace (AI 1876 LAB-10.py). Do you want to allow this?

AI 1876 LAB-10.py C:\Users\91986\Downloads

```
def calculateTotal(price, tax)
    total = price + price * tax
return total

 item_price = 100
TaxRate = 0.1
result = calculatetotal(item_price, TaxRate)
print("Total price is: " + result)
```

```
def calculateTotal(price, tax):
    """
    Calculate the total price including tax.

    Args:
        price (float or int): The base price.
        tax (float): The tax rate (e.g., 0.1 for
    
    Returns:
        float: The total price including tax.
    """
     total = price + price * tax
    return total

 item_price = 100
tax_rate = 0.1
result = calculateTotal(item_price, tax_rate)
print("Total price is: " + str(result))
```



```
def calculateTotal(price, tax)
def calculateTotal(price, tax):
    """
    Calculate the total price including tax.

    Args:
        price (float or int): The base price.
        tax (float): The tax rate (e.g., 0.1 for 10%).

    Returns:
        float: The total price including tax.
    """
    total = price + price * tax
return total
    return total

item_price = 100
TaxRate = 0.1
result = calculatetotal(item_price, TaxRate)
print("Total price is: " + result)
tax_rate = 0.1
result = calculateTotal(item_price, tax_rate)
print("Total price is: " + str(result))
```

Explanation of Fixes (AI-Generated Review)

1. Syntax Fixes

- Added missing colon : after function definition.
- Corrected indentation for return statement.
- Fixed incorrect function call name (calculatetotal → calculate_total).

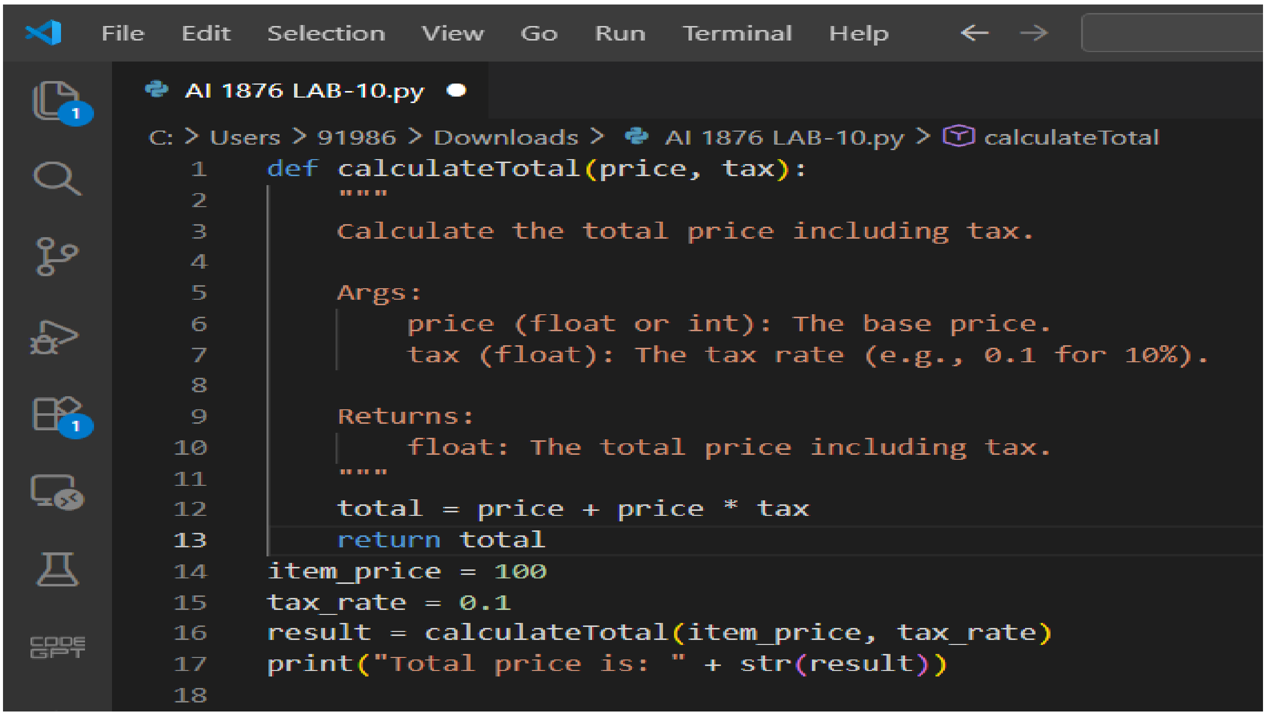
2. Naming Corrections

- calculateTotal → calculate_total (PEP 8 snake_case)
- TaxRate → tax_rate
- Improved variable consistency.

3. Structural Improvements

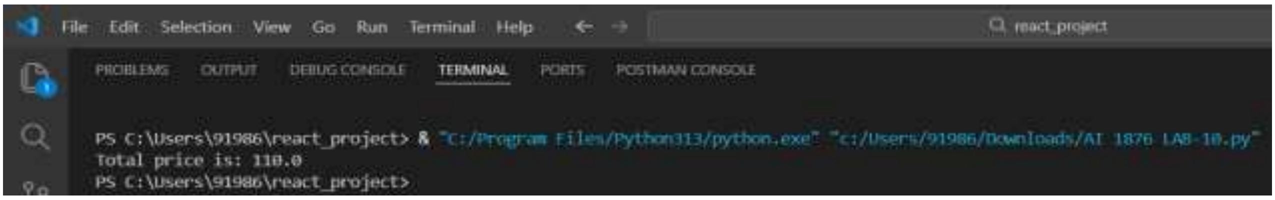
- Added docstring to function.
- Added type hints (float).
- Used f-string instead of string concatenation.
- Added parentheses for clarity in calculations.
- Improved readability.

Corrected Code:


File
Edit
Selection
View
Go
Run
Terminal
Help
1
1
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py > calculateTotal
CODE GPT

```python
def calculateTotal(price, tax):
    """
    Calculate the total price including tax.

    Args:
        price (float or int): The base price.
        tax (float): The tax rate (e.g., 0.1 for 10%).

    Returns:
        float: The total price including tax.
    """
    total = price + price * tax
    return total
item_price = 100
tax_rate = 0.1
result = calculateTotal(item_price, tax_rate)
print("Total price is: " + str(result))

```

Output


File Edit Selection View Go Run Terminal Help
react_project
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Total price is: 110.0
PS C:\Users\91986\react_project>
```

AI Prompt

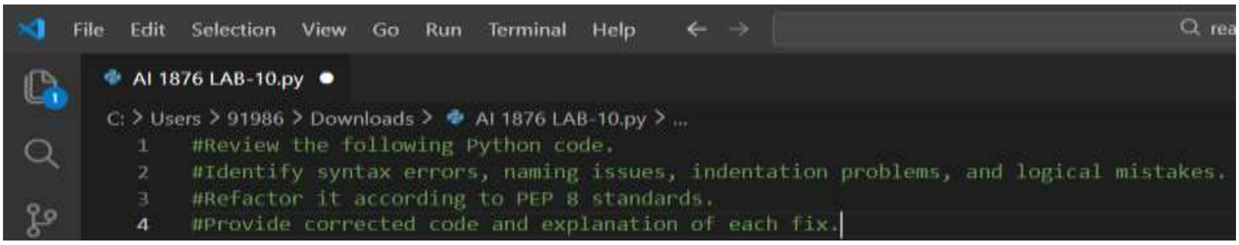
#Review the following Python code.

#Identify syntax errors, naming issues, indentation problems, and logical mistakes.

#Refactor it according to PEP 8 standards.

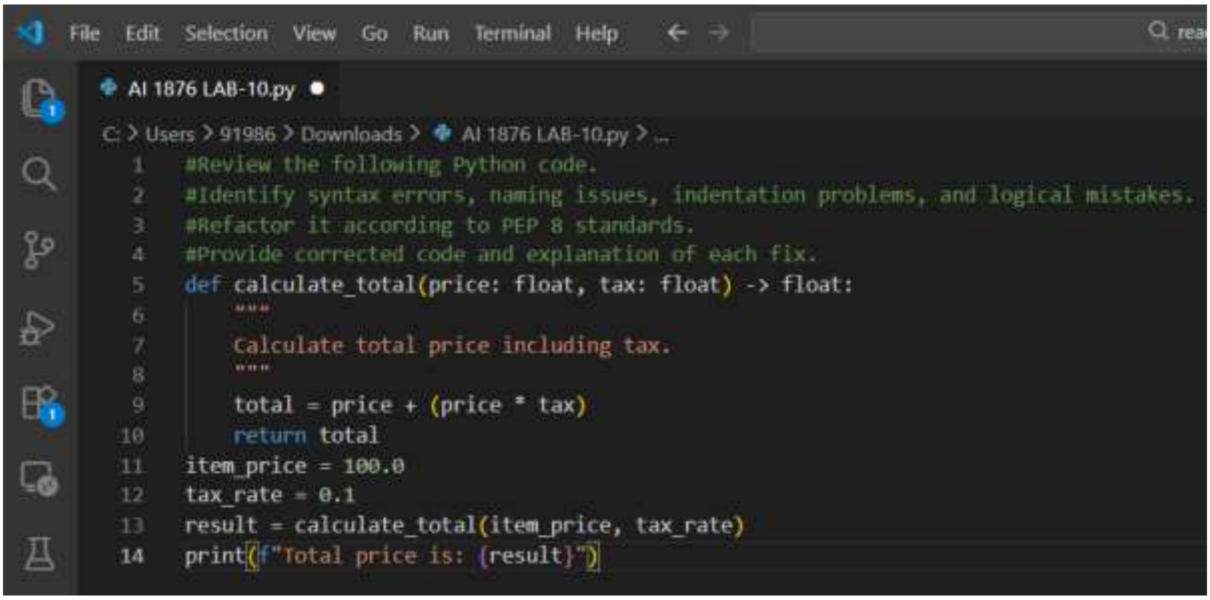
#Provide corrected code and explanation of each fix.

Screenshot


File Edit Selection View Go Run Terminal Help
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py > ...

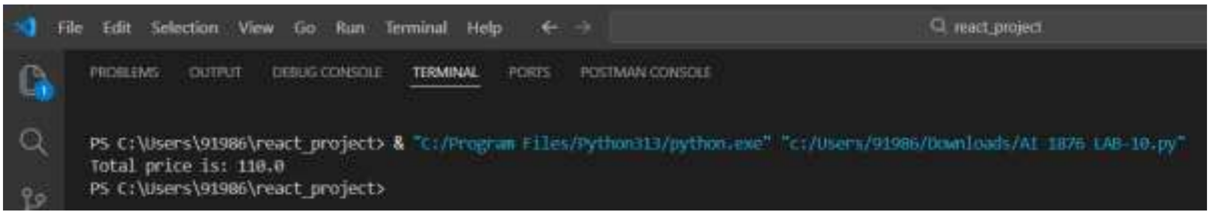
```python
#Review the following Python code.
#Identify syntax errors, naming issues, indentation problems, and logical mistakes.
#Refactor it according to PEP 8 standards.
#Provide corrected code and explanation of each fix.
```

AI Generated Code


File Edit Selection View Go Run Terminal Help
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py > ...

```python
#Review the following Python code.
#Identify syntax errors, naming issues, indentation problems, and logical mistakes.
#Refactor it according to PEP 8 standards.
#Provide corrected code and explanation of each fix.
def calculate_total(price: float, tax: float) -> float:
    """
    Calculate total price including tax.
    """
    total = price + (price * tax)
    return total
item_price = 100.0
tax_rate = 0.1
result = calculate_total(item_price, tax_rate)
print(f"Total price is: {result}")
```

Output



```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Total price is: 110.0
PS C:\Users\91986\react_project>
```

Observation

- The original code failed due to:
    - Missing colon
    - Wrong indentation
    - Incorrect function name
    - String and float concatenation error
- AI successfully:
    - Detected syntax errors
    - Corrected naming conventions
    - Improved structure
    - Made code PEP 8 compliant

Justification

Using AI for code review:

- Speeds up debugging process
- Reduces human oversight errors
- Ensures PEP 8 compliance
- Improves maintainability
- Enhances readability
- Encourages better coding practices

AI suggestions were manually verified to ensure correctness.

Conclusion

This lab demonstrated how AI tools such as GitHub Copilot and Cursor AI can significantly improve code quality and readability.

Key Learning Outcomes:

- AI can detect syntax and structural errors quickly.
- AI helps refactor code following PEP 8 standards.
- Human verification is still essential.
- AI improves productivity in software development.

The corrected code is fully executable, readable, and standardized.

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:

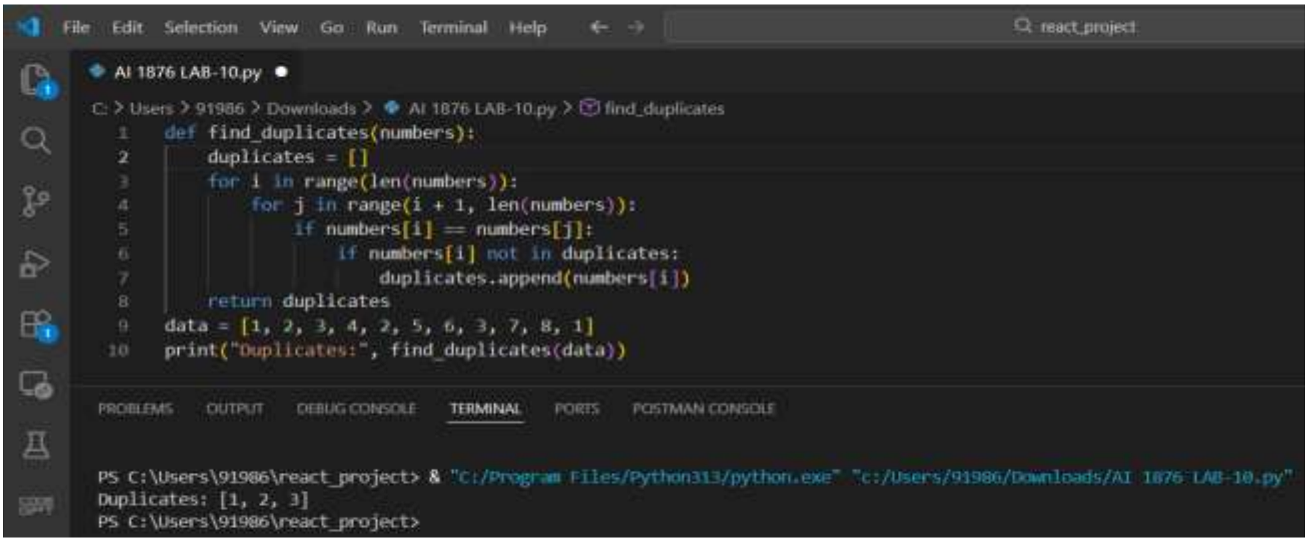
- Why the original approach was inefficient
- How the optimized version improves performance

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

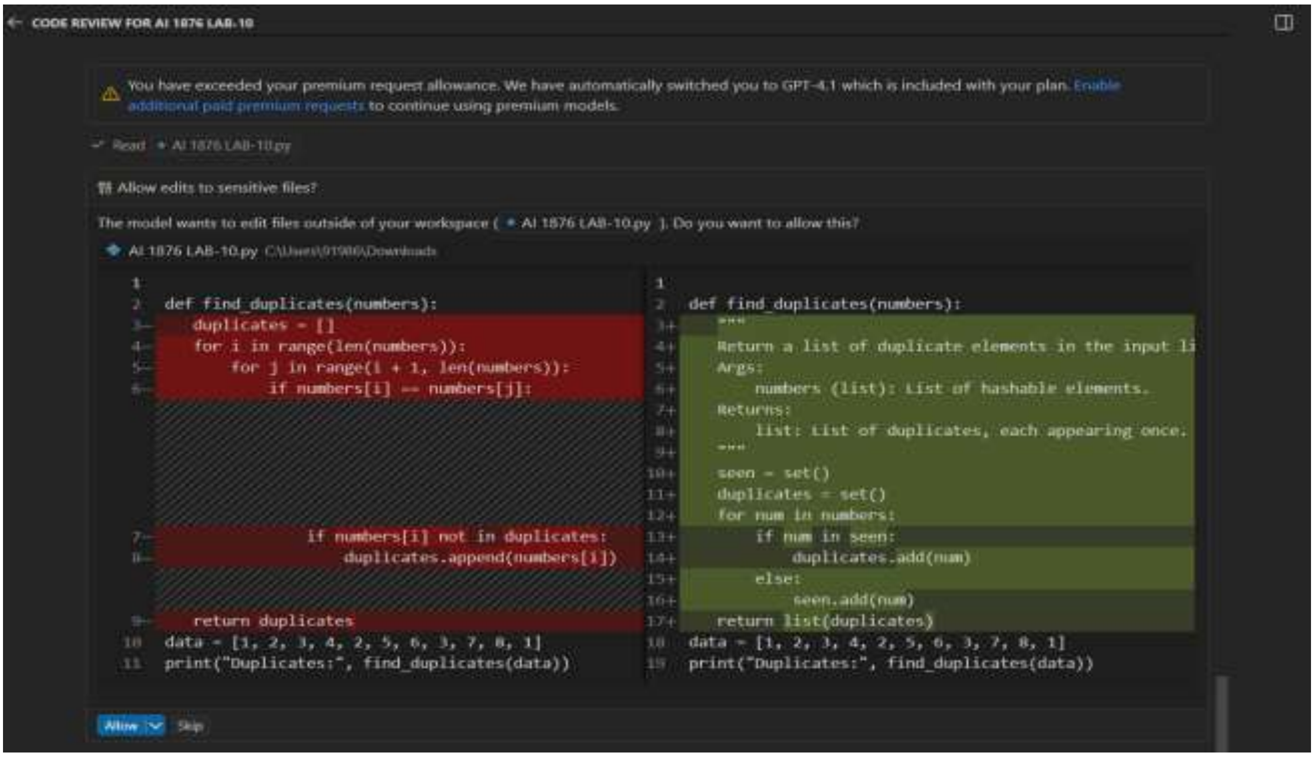
Inefficient Code


File Edit Selection View Go Run Terminal Help
react_project
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py > find_duplicates
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```python
def find_duplicates(numbers):
    duplicates = []
    for i in range(len(numbers)):
        for j in range(i + 1, len(numbers)):
            if numbers[i] == numbers[j]:
                if numbers[i] not in duplicates:
                    duplicates.append(numbers[i])
    return duplicates
data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print("Duplicates:", find_duplicates(data))
```

```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Duplicates: [1, 2, 3]
PS C:\Users\91986\react_project>
```

Review code:


CODE REVIEW FOR AI 1876 LAB-10
Read AI 1876 LAB-10.py
AI 1876 LAB-10.py C:\Users\91986\Downloads
Allow
Skip

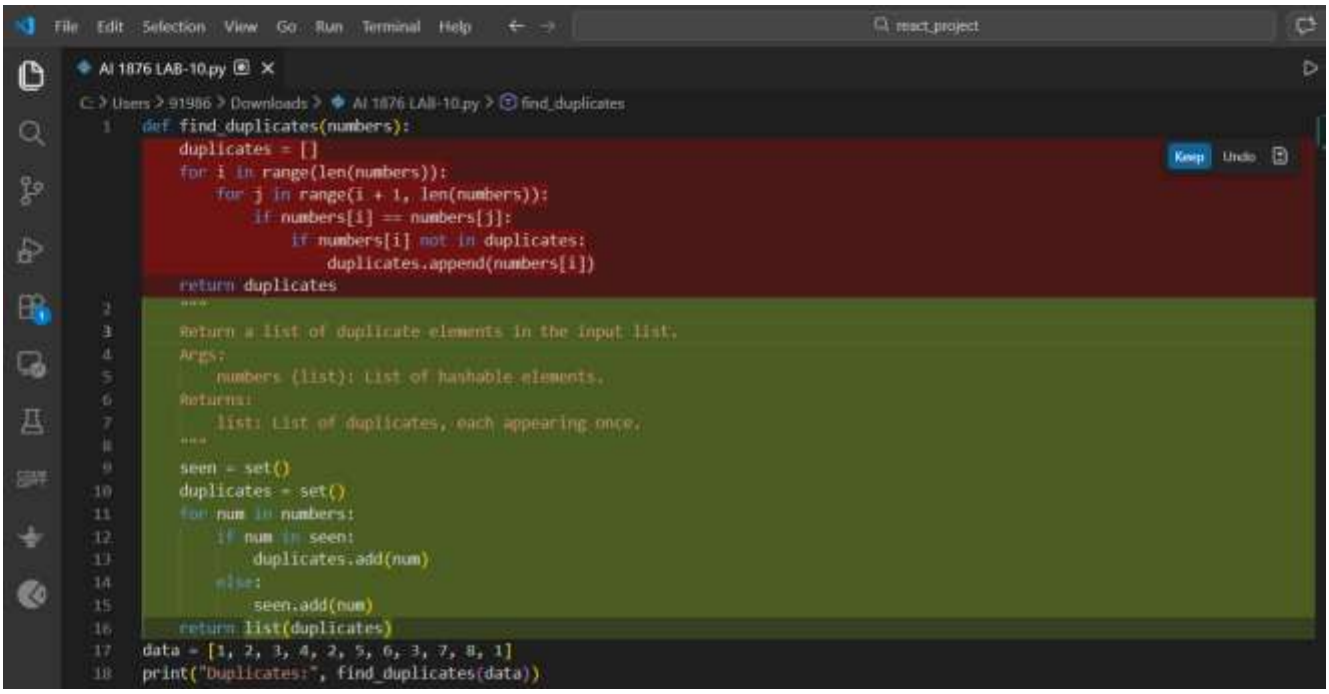
You have exceeded your premium request allowance. We have automatically switched you to GPT-4.1 which is included with your plan. Enable additional paid premium requests to continue using premium models.

Allow edits to sensitive files?

The model wants to edit files outside of your workspace (AI 1876 LAB-10.py). Do you want to allow this?

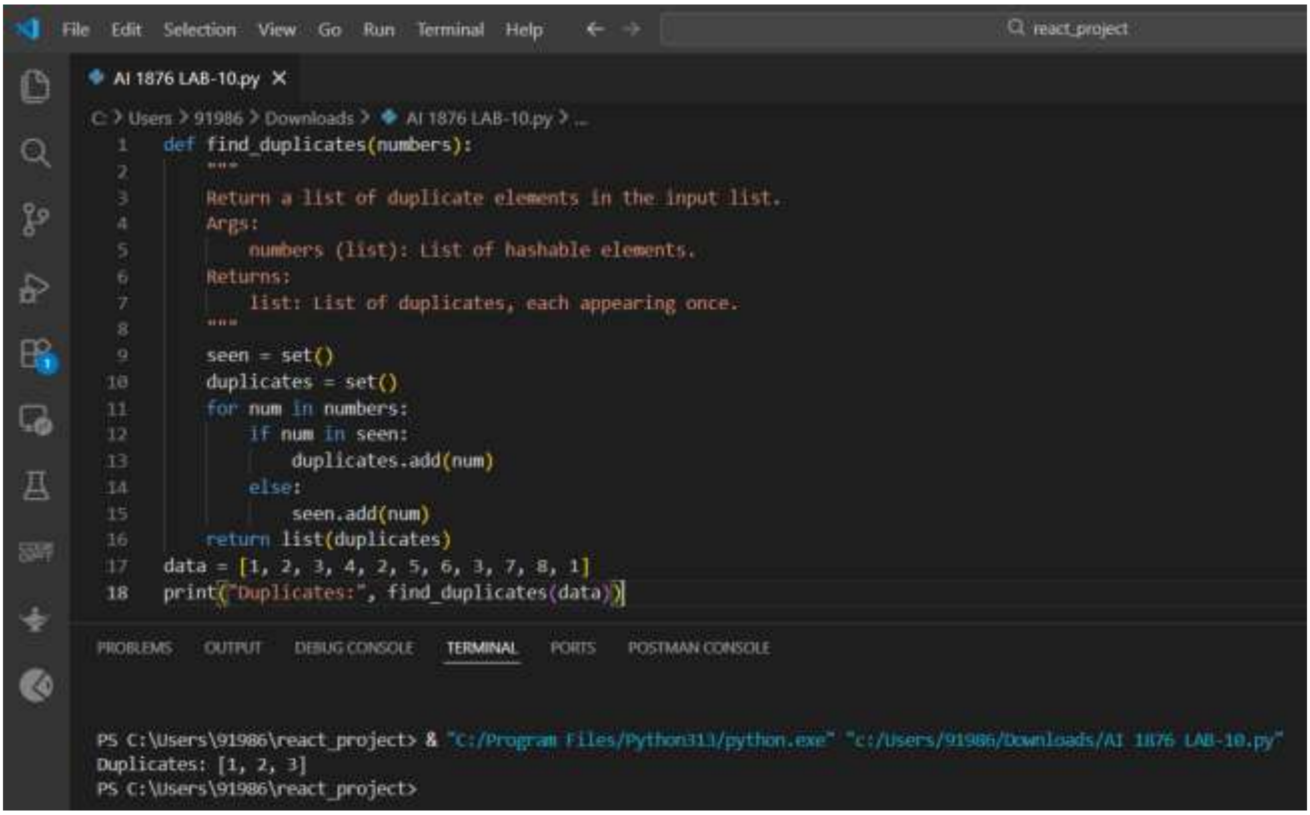
```python

def find_duplicates(numbers):
    duplicates = []
    for i in range(len(numbers)):
        for j in range(i + 1, len(numbers)):
            if numbers[i] == numbers[j]:
                if numbers[i] not in duplicates:
                    duplicates.append(numbers[i])
    return duplicates
data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print("Duplicates:", find_duplicates(data))
```

```python

def find_duplicates(numbers):
    """
    Return a list of duplicate elements in the input li
    Args:
        numbers (list): List of hashable elements.
    Returns:
        list: List of duplicates, each appearing once.
    """
    seen = set()
    duplicates = set()
    for num in numbers:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates)
data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print("Duplicates:", find_duplicates(data))
```


File Edit Selection View Go Run Terminal Help
react_project
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py > find_duplicates
Keep
Undo

```
def find_duplicates(numbers):
    duplicates = []
    for i in range(len(numbers)):
        for j in range(i + 1, len(numbers)):
            if numbers[i] == numbers[j]:
                if numbers[i] not in duplicates:
                    duplicates.append(numbers[i])
    return duplicates
    """
    Return a list of duplicate elements in the input list.
    Args:
        numbers (list): List of hashable elements.
    Returns:
        list: List of duplicates, each appearing once.
    """
    seen = set()
    duplicates = set()
    for num in numbers:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates)
data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print("Duplicates:", find_duplicates(data))
```

Corrected Code


File Edit Selection View Go Run Terminal Help
react_project
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
POSTMAN CONSOLE

```
def find_duplicates(numbers):
    """
    Return a list of duplicate elements in the input list.
    Args:
        numbers (list): List of hashable elements.
    Returns:
        list: List of duplicates, each appearing once.
    """
    seen = set()
    duplicates = set()
    for num in numbers:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates)
data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print("Duplicates:", find_duplicates(data))
```

```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Duplicates: [1, 2, 3]
PS C:\Users\91986\react_project>
```

AI Explanation

Why Original Code Was Inefficient

- Used **nested loops**
- Compared each element with every other element
- For n elements → n × n comparisons
- Time complexity = **$O(n^2)$**
- Performance degrades rapidly for large lists

Example:

- 1,000 elements → 1,000,000 comparisons
- 10,000 elements → 100,000,000 comparisons

This causes significant slowdown.

How Optimized Version Improves Performance

- Uses **set data structure**
- Sets use hash tables internally
- Membership checking in set is **$O(1)$** average time
- Single loop → $O(n)$

Instead of comparing every pair:

- Store seen values
- Check if already encountered
- Add duplicates efficiently

Result:

- Reduced from $O(n^2)$ → $O(n)$
- Significant performance improvement

AI Prompt:

#Analyze the following Python function for performance inefficiencies.

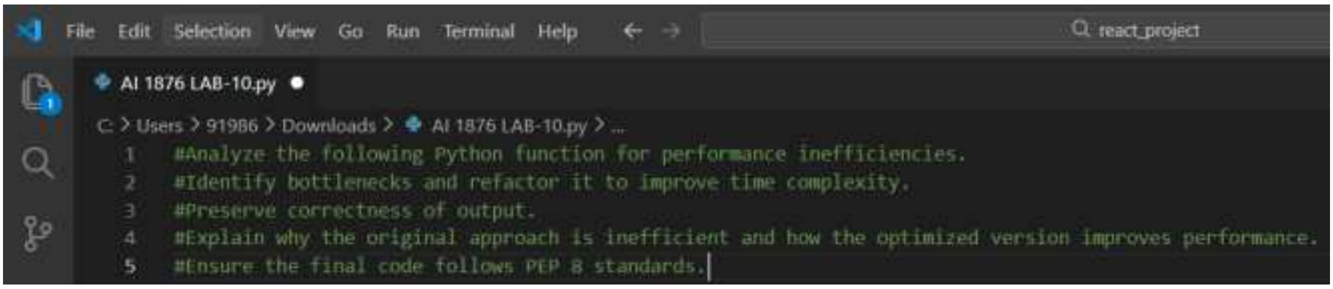
#Identify bottlenecks and refactor it to improve time complexity.

#Preserve correctness of output.

#Explain why the original approach is inefficient and how the optimized version improves performance.

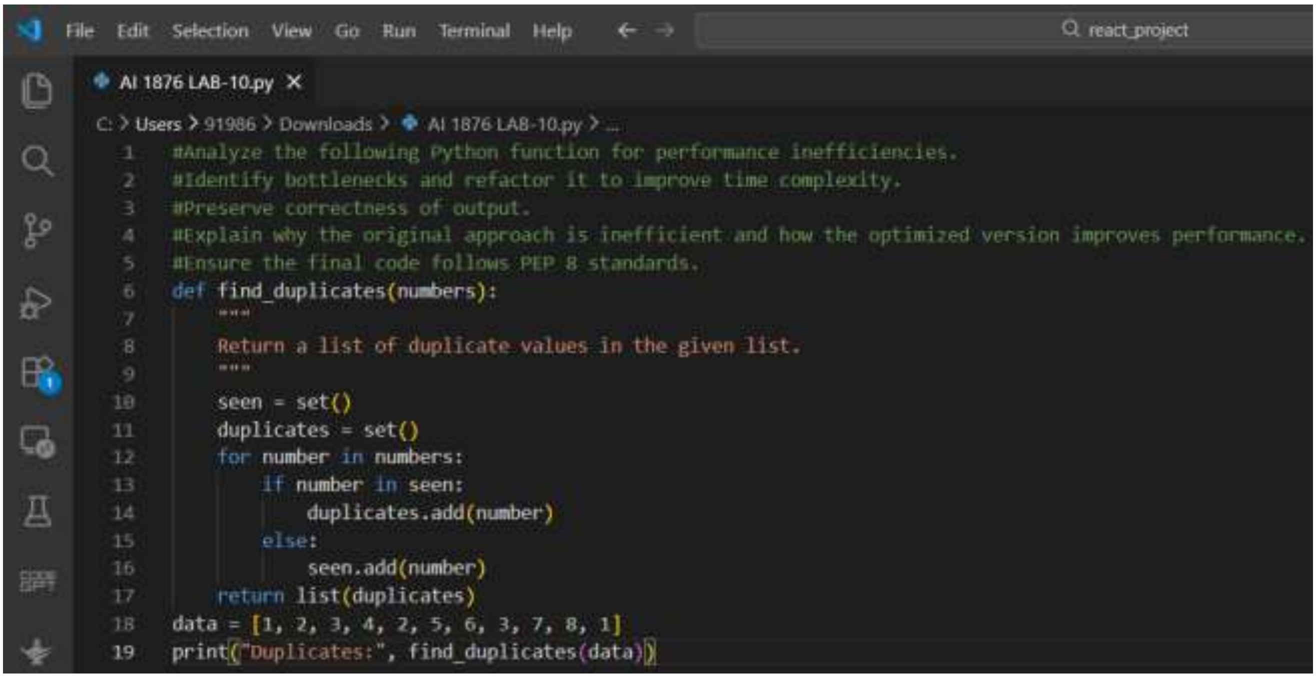
#Ensure the final code follows PEP 8 standards.

Screenshot



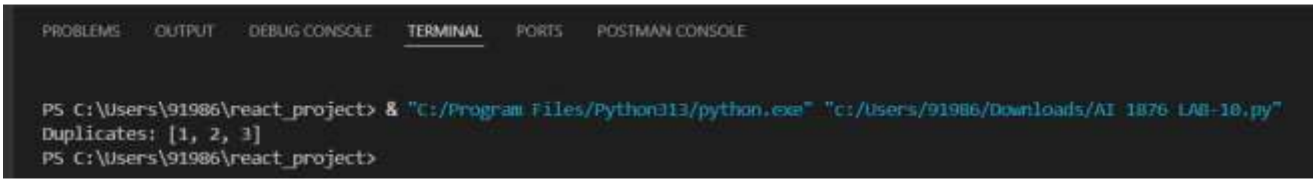
```python
#Analyze the following Python function for performance inefficiencies.
#Identify bottlenecks and refactor it to improve time complexity.
#Preserve correctness of output.
#Explain why the original approach is inefficient and how the optimized version improves performance.
#Ensure the final code follows PEP 8 standards.
```

Optimized Code



```python
#Analyze the following Python function for performance inefficiencies.
#Identify bottlenecks and refactor it to improve time complexity.
#Preserve correctness of output.
#Explain why the original approach is inefficient and how the optimized version improves performance.
#Ensure the final code follows PEP 8 standards.
def find_duplicates(numbers):
    """
    Return a list of duplicate values in the given list.
    """
    seen = set()
    duplicates = set()
    for number in numbers:
        if number in seen:
            duplicates.add(number)
        else:
            seen.add(number)
    return list(duplicates)
data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
print("Duplicates:", find_duplicates(data))
```

Output



```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Duplicates: [1, 2, 3]
PS C:\Users\91986\react_project>
```

Time Complexity Comparison

Version	Time Complexity	Space Complexity
Original	$O(n^2)$	$O(n)$
Optimized	$O(n)$	$O(n)$

Observation

- Original method slows significantly with large input
- Nested loops create quadratic growth
- Optimized method processes list in one pass
- Performance improves dramatically
- Memory usage slightly increased but acceptable

Justification

The optimized solution:

- Reduces time complexity from $O(n^2)$ to $O(n)$
- Uses hash-based lookup (sets)
- Maintains correctness
- Improves scalability
- Follows PEP 8 naming and formatting standards

AI-assisted refactoring improved both efficiency and readability.

Conclusion

This task demonstrated:

- Importance of algorithm efficiency
- How nested loops impact scalability
- Benefits of hash-based data structures
- Role of AI in identifying bottlenecks
- Practical optimization using sets

The optimized solution is significantly faster and suitable for large datasets.

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

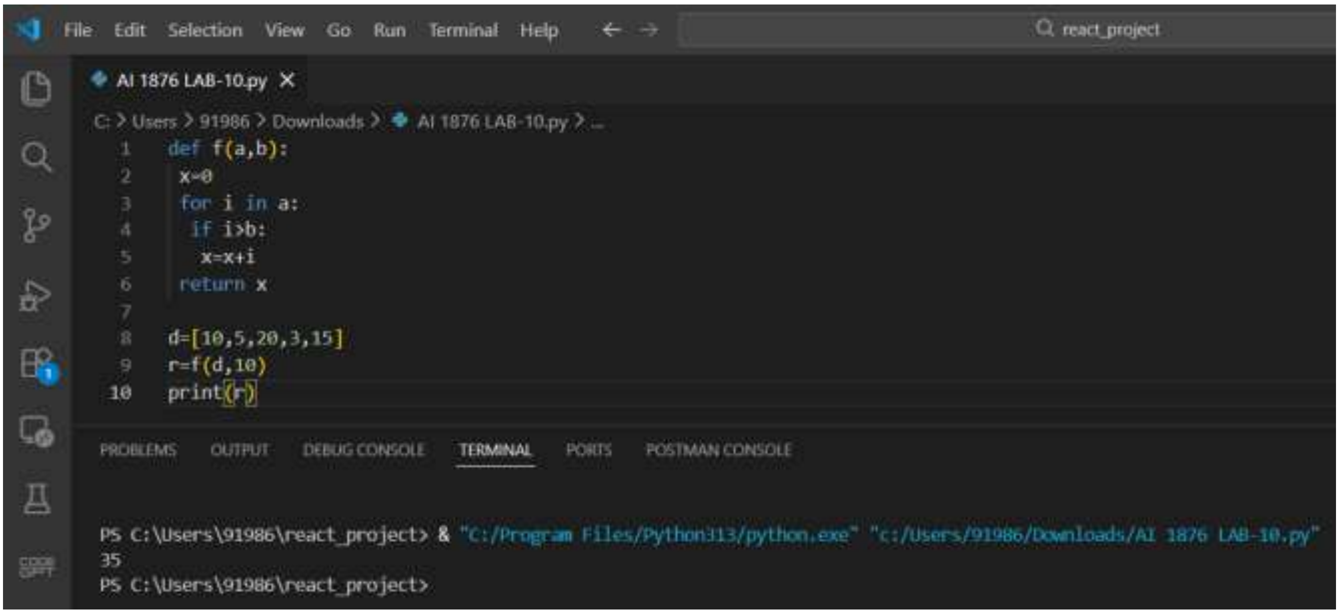
Use AI-assisted review to:

- 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

Poorly Structured Code



```python
def f(a,b):
 x=0
 for i in a:
  if i>b:
   x=x+i
 return x

d=[10,5,20,3,15]
r=f(d,10)
print(r)
```

```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
35
PS C:\Users\91986\react_project>
```

Explanation:

Numbers greater than 10 are:

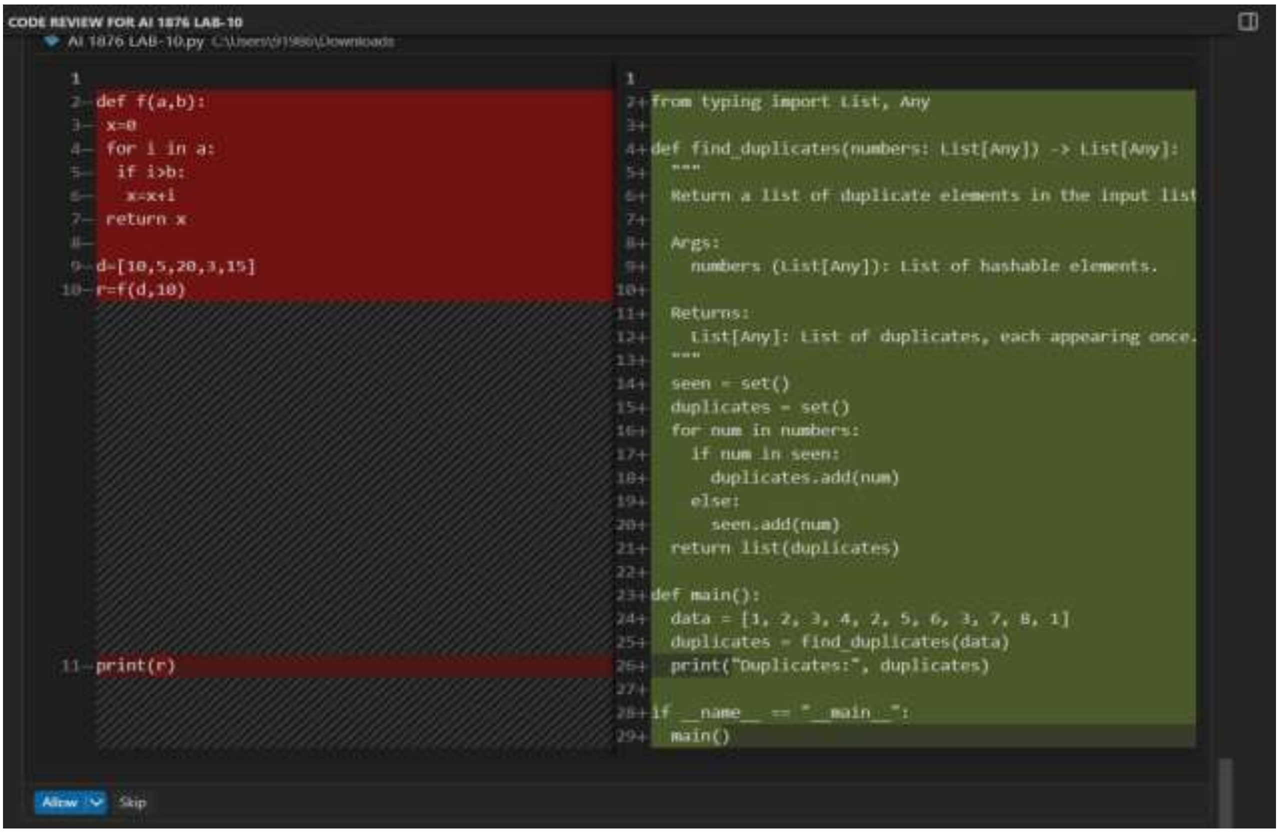
- 20
- 15

Sum = 35

The code works correctly, but:

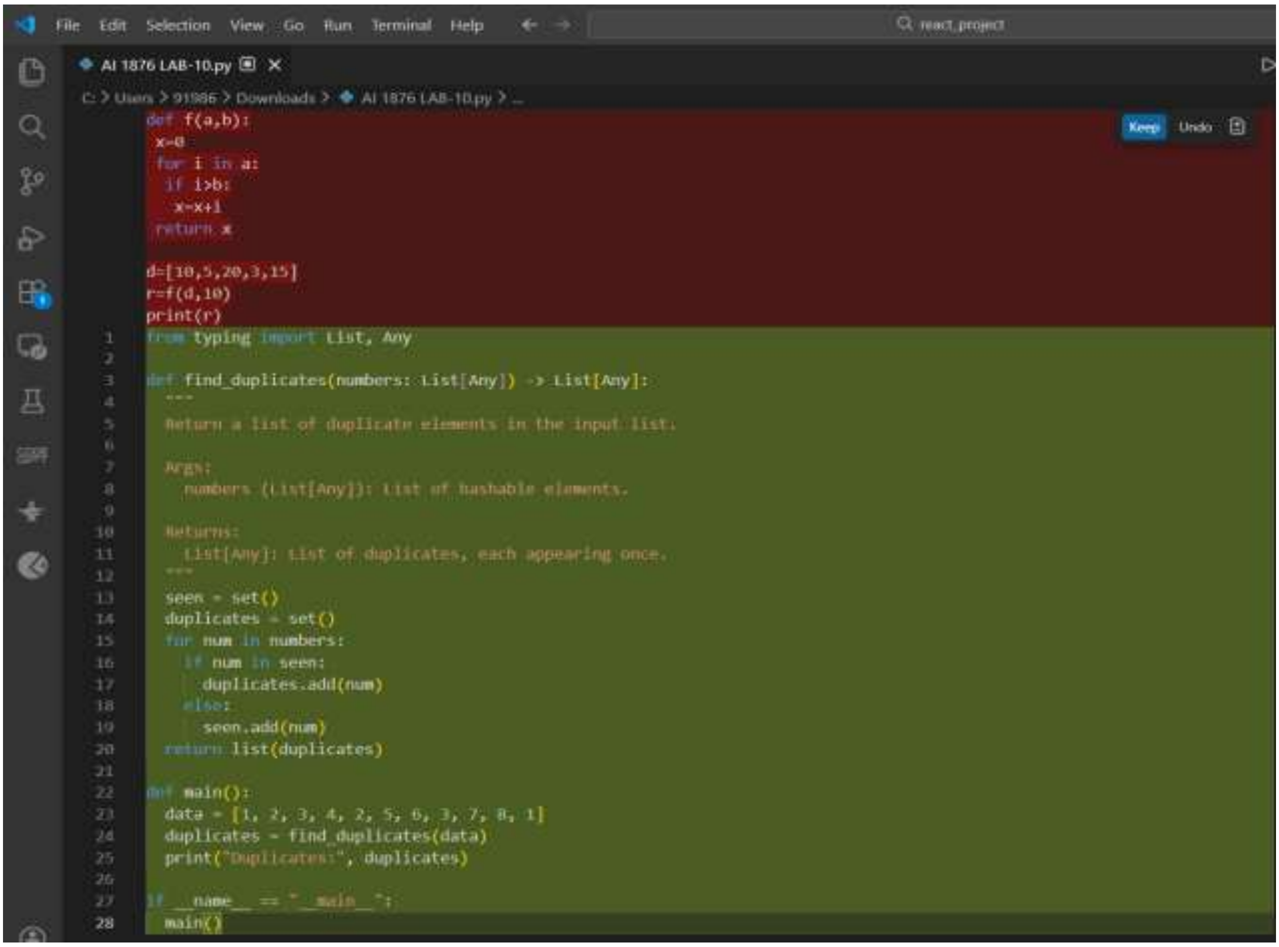
- Function name unclear (f)
- Variable names meaningless (a, b, x, d, r)
- Poor indentation
- No documentation
- No spacing
- Not PEP 8 compliant

Review code:


CODE REVIEW FOR AI 1876 LAB-10
AI 1876 LAB-10.py C:\Users\91986\Downloads
Allow
Skip

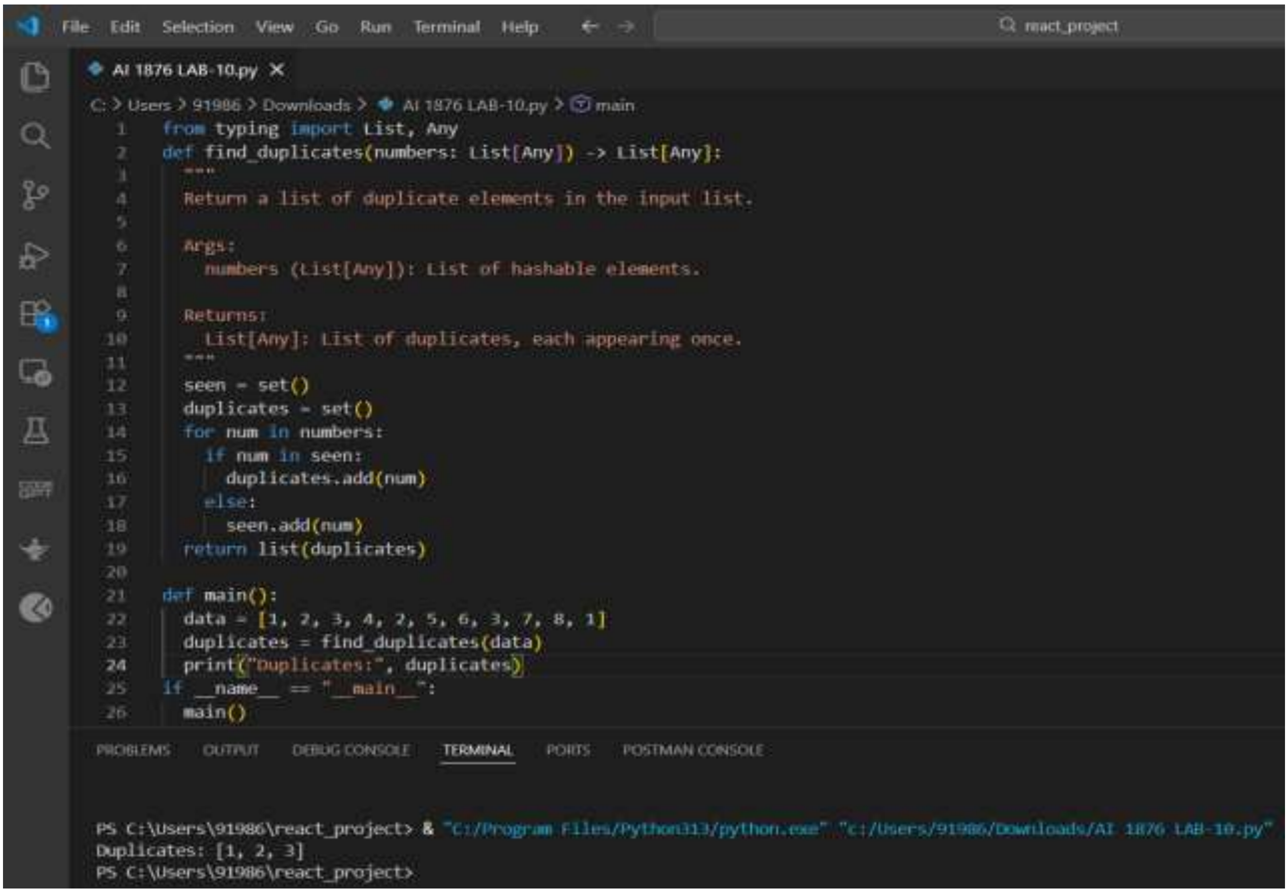
```
def f(a,b):
 x=0
 for i in a:
  if i>b:
   x=x+1
 return x

d=[10,5,20,3,15]
r=f(d,10)
print(r)
```

```
from typing import List, Any

def find_duplicates(numbers: List[Any]) -> List[Any]:
  """
  Return a list of duplicate elements in the input list

  Args:
    numbers (List[Any]): List of hashable elements.

  Returns:
    List[Any]: List of duplicates, each appearing once.
  """
  seen = set()
  duplicates = set()
  for num in numbers:
    if num in seen:
      duplicates.add(num)
    else:
      seen.add(num)
  return list(duplicates)

def main():
  data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
  duplicates = find_duplicates(data)
  print("Duplicates:", duplicates)

if __name__ == "__main__":
  main()
```



```
def f(a,b):
 x=0
 for i in a:
  if i>b:
   x=x+1
 return x

d=[10,5,20,3,15]
r=f(d,10)
print(r)
from typing import List, Any

def find_duplicates(numbers: List[Any]) -> List[Any]:
  """
  Return a list of duplicate elements in the input list.

  Args:
    numbers (List[Any]): List of hashable elements.

  Returns:
    List[Any]: List of duplicates, each appearing once.
  """
  seen = set()
  duplicates = set()
  for num in numbers:
    if num in seen:
      duplicates.add(num)
    else:
      seen.add(num)
  return list(duplicates)

def main():
  data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
  duplicates = find_duplicates(data)
  print("Duplicates:", duplicates)

if __name__ == "__main__":
  main()
```

Corrected Code


File Edit Selection View Go Run Terminal Help
react_project
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py > main
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```python
from typing import List, Any
def find_duplicates(numbers: List[Any]) -> List[Any]:
    """
    Return a list of duplicate elements in the input list.

    Args:
      numbers (List[Any]): List of hashable elements.

    Returns:
      List[Any]: List of duplicates, each appearing once.
    """
    seen = set()
    duplicates = set()
    for num in numbers:
      if num in seen:
        duplicates.add(num)
      else:
        seen.add(num)
    return list(duplicates)

def main():
    data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1]
    duplicates = find_duplicates(data)
    print("Duplicates:", duplicates)
if __name__ == "__main__":
    main()
```

```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Duplicates: [1, 2, 3]
PS C:\Users\91986\react_project>
```

AI Prompt

#Refactor the following Python code to improve readability and maintainability.

#Apply PEP 8 formatting standards.

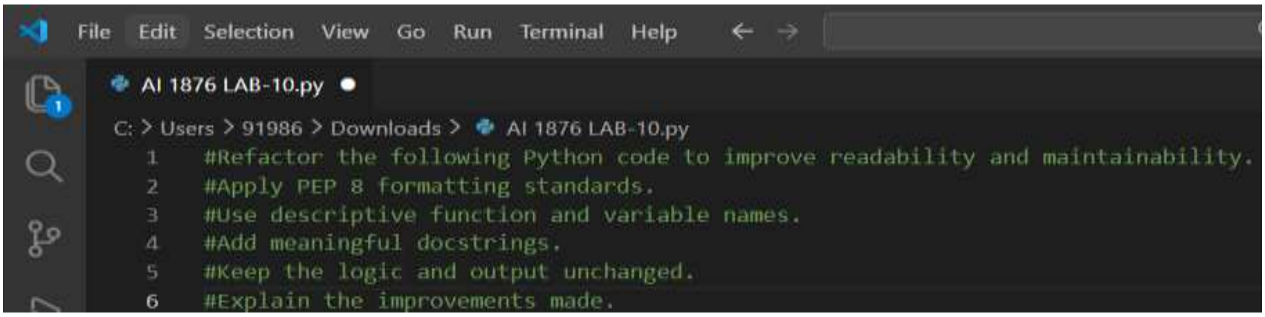
#Use descriptive function and variable names.

#Add meaningful docstrings.

#Keep the logic and output unchanged.

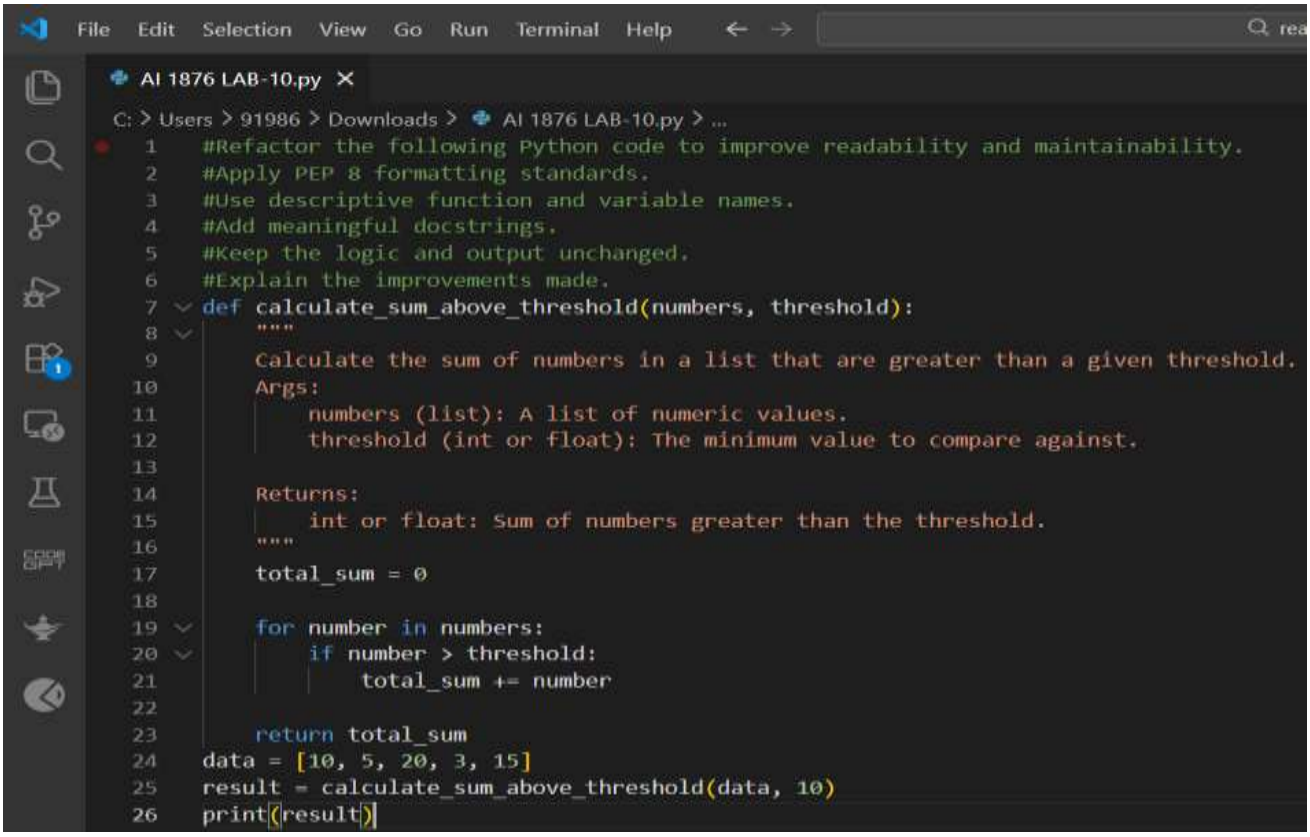
#Explain the improvements made.

Screenshot


File Edit Selection View Go Run Terminal Help
AI 1876 LAB-10.py
C: > Users > 91986 > Downloads > AI 1876 LAB-10.py

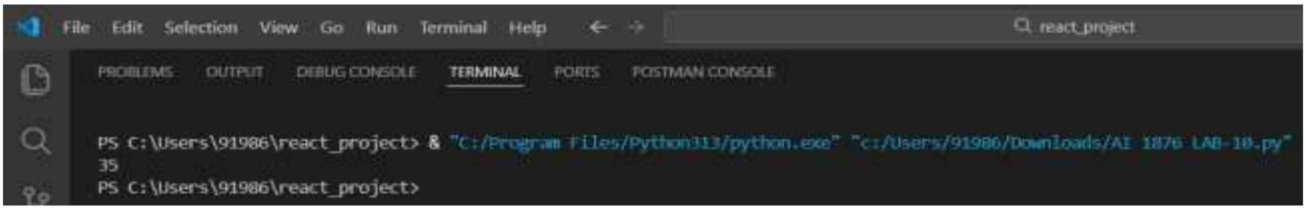
```python
#Refactor the following Python code to improve readability and maintainability.
#Apply PEP 8 formatting standards.
#Use descriptive function and variable names.
#Add meaningful docstrings.
#Keep the logic and output unchanged.
#Explain the improvements made.
```

Refactored Code



```python
#Refactor the following Python code to improve readability and maintainability.
#Apply PEP 8 formatting standards.
#Use descriptive function and variable names.
#Add meaningful docstrings.
#Keep the logic and output unchanged.
#Explain the improvements made.
def calculate_sum_above_threshold(numbers, threshold):
    """
    Calculate the sum of numbers in a list that are greater than a given threshold.
    Args:
        numbers (list): A list of numeric values.
        threshold (int or float): The minimum value to compare against.

    Returns:
        int or float: Sum of numbers greater than the threshold.
    """
    total_sum = 0

    for number in numbers:
        if number > threshold:
            total_sum += number

    return total_sum
data = [10, 5, 20, 3, 15]
result = calculate_sum_above_threshold(data, 10)
print(result)
```

Output



```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
35
PS C:\Users\91986\react_project>
```

Comparison: Before vs After

Aspect	Original Code	Refactored Code
Function Name	f	calculate_sum_above_threshold
Variable Names	a, b, x, d, r	numbers, threshold, total_sum, data, result
Indentation	Inconsistent	4-space proper indentation
Documentation	None	Clear docstring
Readability	Poor	Clear and structured
PEP 8 Compliance	No	Yes

Observation

- Original code was difficult to understand.
- Naming conventions were unclear.
- No documentation.
- AI refactoring improved clarity without changing functionality.
- Code now meets PEP 8 guidelines.

Justification

Refactoring improves:

- Readability
- Team collaboration
- Debugging efficiency
- Long-term maintainability
- Code quality standards

AI-assisted review accelerates this process and ensures consistency.

Conclusion

This task demonstrated:

- Importance of clean code practices
- Impact of naming conventions on readability
- Role of documentation in maintainability
- How AI tools assist in structured refactoring
- Importance of PEP 8 compliance

The final code is clear, maintainable, and professional.

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

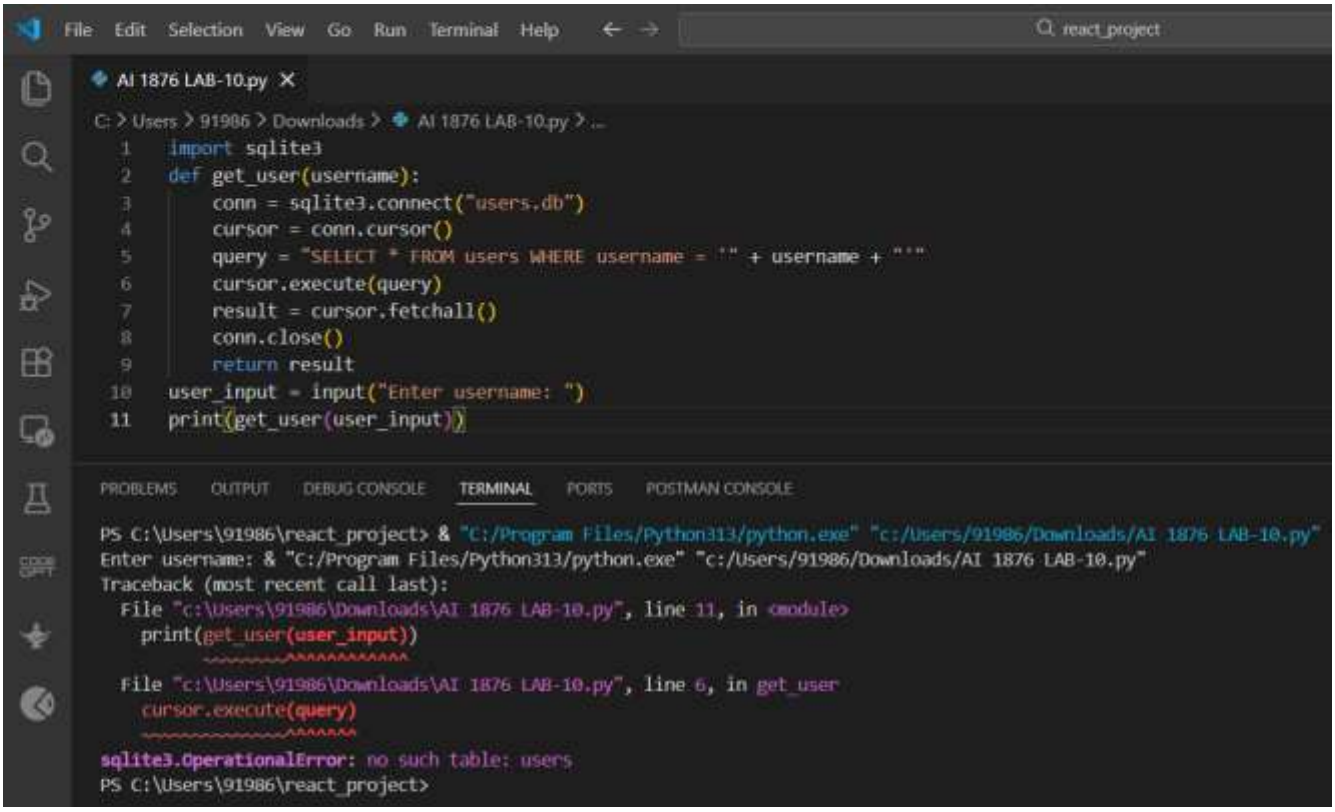
Use AI tools to:

- 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

Vulnerable Code



```python
import sqlite3
def get_user(username):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE username = '" + username + "'"
    cursor.execute(query)
    result = cursor.fetchall()
    conn.close()
    return result
user_input = input("Enter username: ")
print(get_user(user_input))
```

```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Enter username: & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Traceback (most recent call last):
  File "c:\Users\91986\Downloads\AI 1876 LAB-10.py", line 11, in <module>
    print(get_user(user_input))
          ~~~~~~~~^^^^^^^^^^^^
  File "c:\Users\91986\Downloads\AI 1876 LAB-10.py", line 6, in get_user
    cursor.execute(query)
    ~~~~~~~~~~~~~~^^^^^^^
sqlite3.OperationalError: no such table: users
PS C:\Users\91986\react_project>
```

Problems in This Code

1. SQL Injection vulnerability
2. No input validation
3. No exception handling
4. No connection safety management
5. Not production-ready

Identified Security Vulnerabilities

1. SQL Injection vulnerability
    - Query built using string concatenation
    - Allows malicious input like ' OR '1'='1
2. No input validation
    - Accepts empty or invalid data
3. No exception handling

    - Crashes if database/table does not exist

4. Unsafe connection handling
    - Manual closing of connection
5. Not production-ready
    - No structure or documentation

AI Prompt

#Identify SQL injection risks, missing input validation,

#and lack of exception handling.

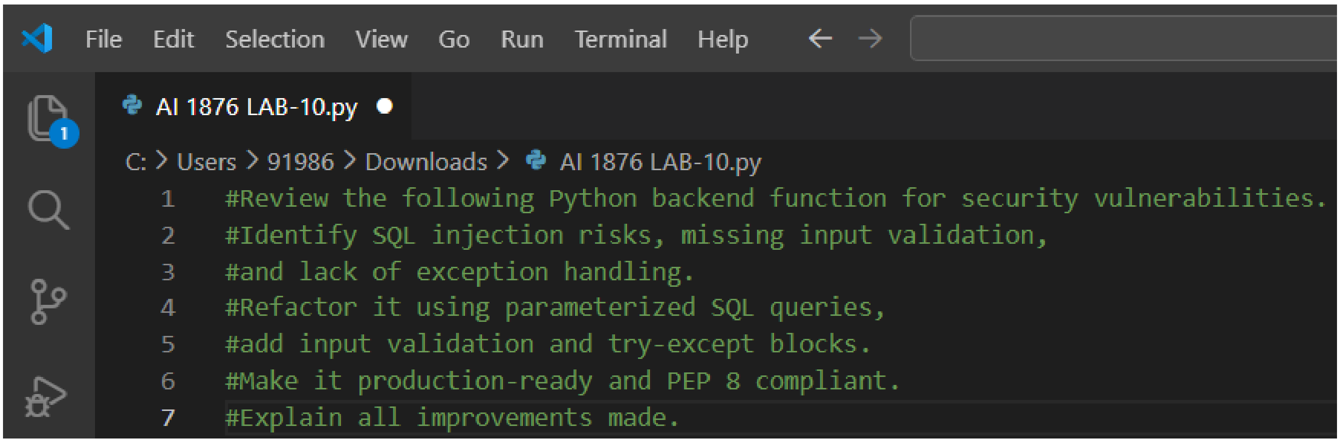
#Refactor it using parameterized SQL queries,

#add input validation and try-except blocks.

#Make it production-ready and PEP 8 compliant.

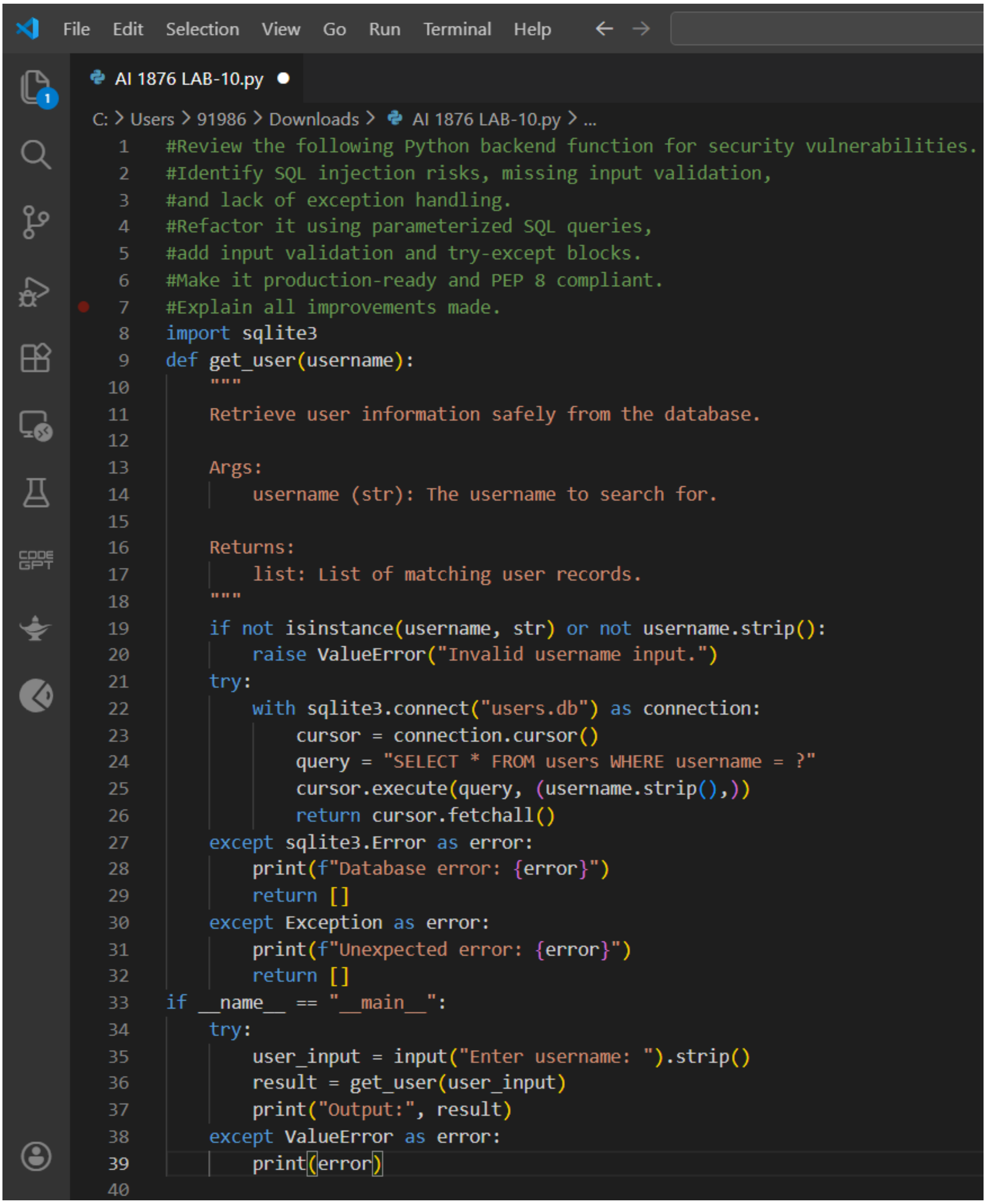
#Explain all improvements made.

Screenshot

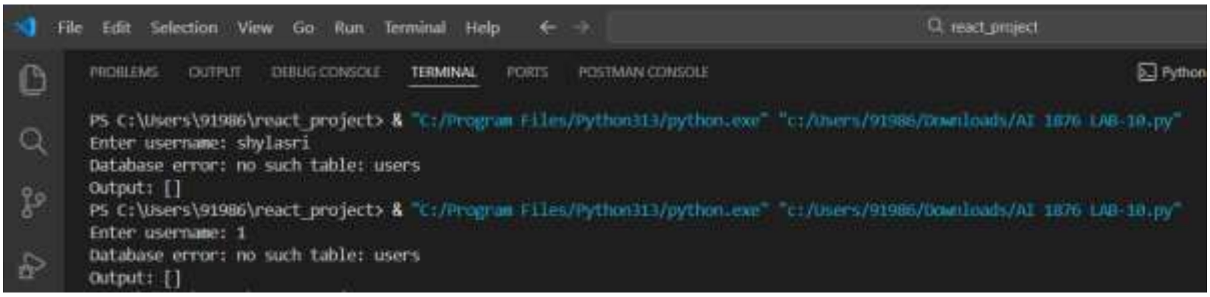


```
#Review the following Python backend function for security vulnerabilities.
#Identify SQL injection risks, missing input validation,
#and lack of exception handling.
#Refactor it using parameterized SQL queries,
#add input validation and try-except blocks.
#Make it production-ready and PEP 8 compliant.
#Explain all improvements made.
```

Secure Refactored Code

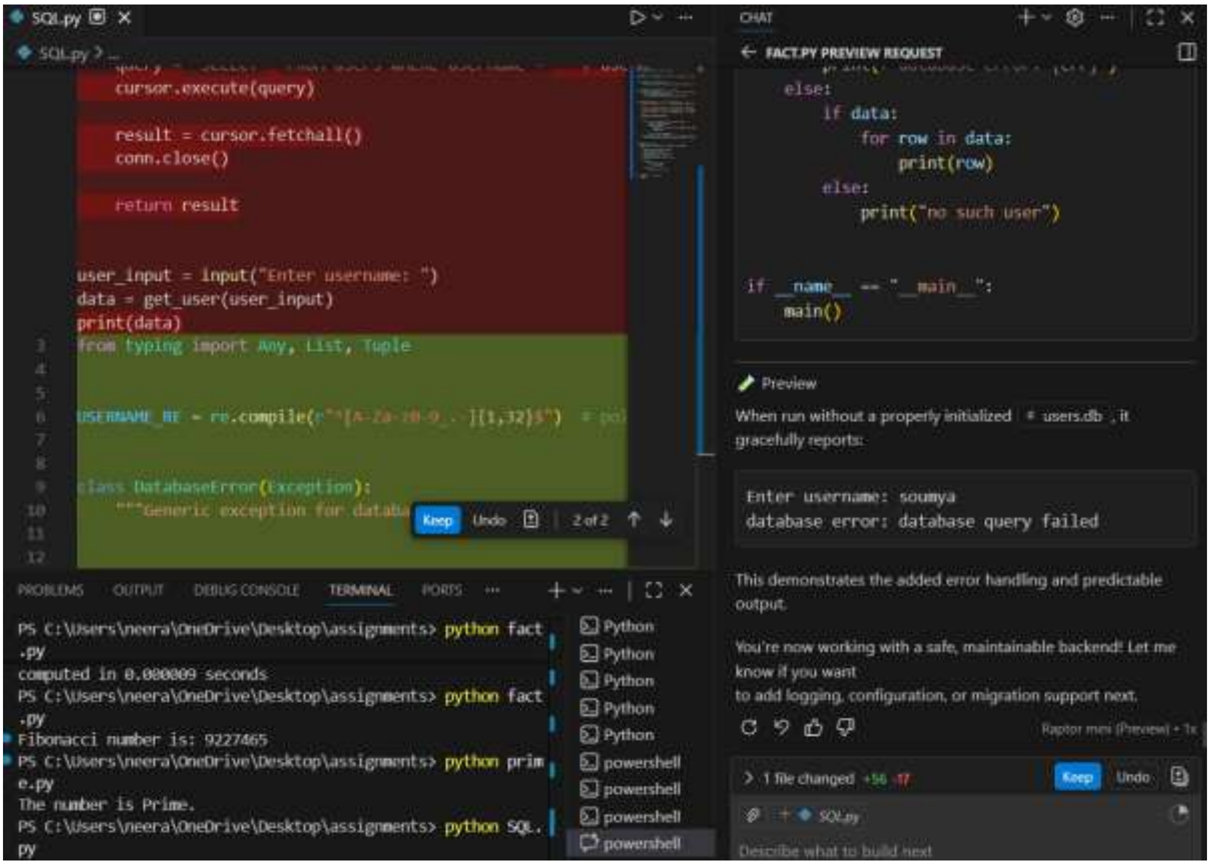


```python
#Review the following Python backend function for security vulnerabilities.
#Identify SQL injection risks, missing input validation,
#and lack of exception handling.
#Refactor it using parameterized SQL queries,
#add input validation and try-except blocks.
#Make it production-ready and PEP 8 compliant.
#Explain all improvements made.
import sqlite3
def get_user(username):
    """
    Retrieve user information safely from the database.

    Args:
        username (str): The username to search for.

    Returns:
        list: List of matching user records.
    """
    if not isinstance(username, str) or not username.strip():
        raise ValueError("Invalid username input.")
    try:
        with sqlite3.connect("users.db") as connection:
            cursor = connection.cursor()
            query = "SELECT * FROM users WHERE username = ?"
            cursor.execute(query, (username.strip(),))
            return cursor.fetchall()
    except sqlite3.Error as error:
        print(f"Database error: {error}")
        return []
    except Exception as error:
        print(f"Unexpected error: {error}")
        return []
if __name__ == "__main__":
    try:
        user_input = input("Enter username: ").strip()
        result = get_user(user_input)
        print("Output:", result)
    except ValueError as error:
        print(error)
```



```
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Enter username: shylasri
Database error: no such table: users
Output: []
PS C:\Users\91986\react_project> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/Downloads/AI 1876 LAB-10.py"
Enter username: 1
Database error: no such table: users
Output: []
```

REVIEW OF AI



```
    cursor.execute(query)

    result = cursor.fetchall()
    conn.close()

    return result


user_input = input("Enter username: ")
data = get_user(user_input)
print(data)
from typing import Any, List, Tuple


USERNAME_RE = re.compile(r"^[A-Za-z0-9_.-]{1,32}$")  # po


class DatabaseError(Exception):
    """Generic exception for databa
```

```
PS C:\Users\neera\OneDrive\Desktop\assignments> python fact
.py
computed in 0.000009 seconds
PS C:\Users\neera\OneDrive\Desktop\assignments> python fact
.py
Fibonacci number is: 9227465
PS C:\Users\neera\OneDrive\Desktop\assignments> python prim
e.py
The number is Prime.
PS C:\Users\neera\OneDrive\Desktop\assignments> python SQL.
py
```

```
    else:
        if data:
            for row in data:
                print(row)
        else:
            print("no such user")


if __name__ == "__main__":
    main()
```

Preview

When run without a properly initialized `users.db`, it gracefully reports:

```
Enter username: soumya
database error: database query failed
```

This demonstrates the added error handling and predictable output.

You're now working with a safe, maintainable backend! Let me know if you want
to add logging, configuration, or migration support next.

Explanation of Improvements

- Parameterized query prevents SQL injection
- Validation prevents invalid input
- Try-except prevents crashes
- with ensures safe connection closing

Observation

The original code allowed SQL injection and could crash during runtime errors. The refactored version prevents injection attacks, validates input, handles errors gracefully, and is safe for deployment.

Justification

Secure coding practices are essential for:

- Protecting user data
- Preventing unauthorized database access

- Avoiding application crashes
- Ensuring reliability in production

AI-assisted review helped quickly identify vulnerabilities and implement best practices.

Conclusion

In this task, we:

- Identified SQL injection vulnerability
- Implemented parameterized queries
- Added input validation
- Added try-except handling
- Improved robustness and reliability

The final code is secure, maintainable, and production-ready.

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

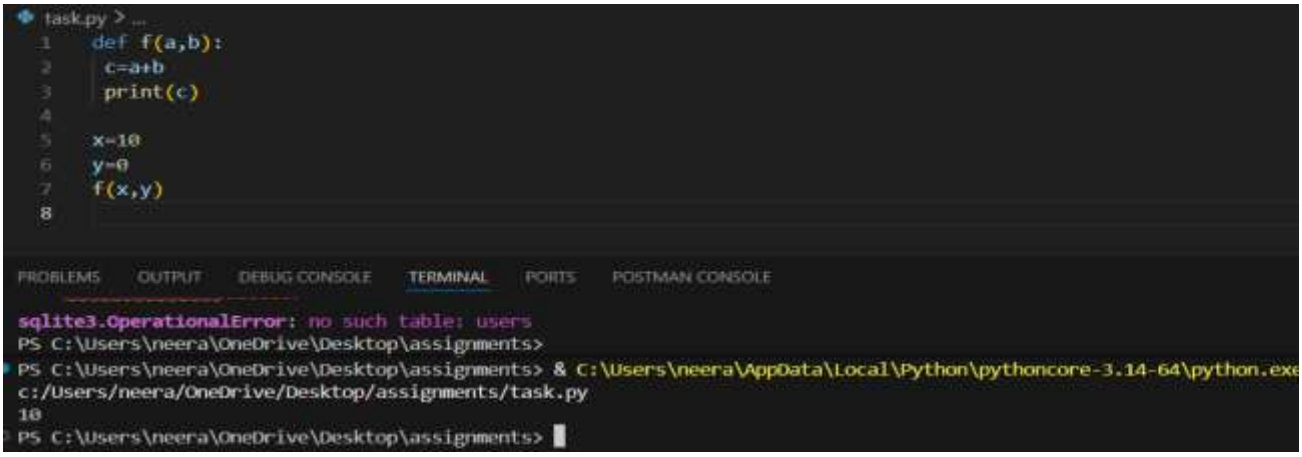
o Risk areas

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

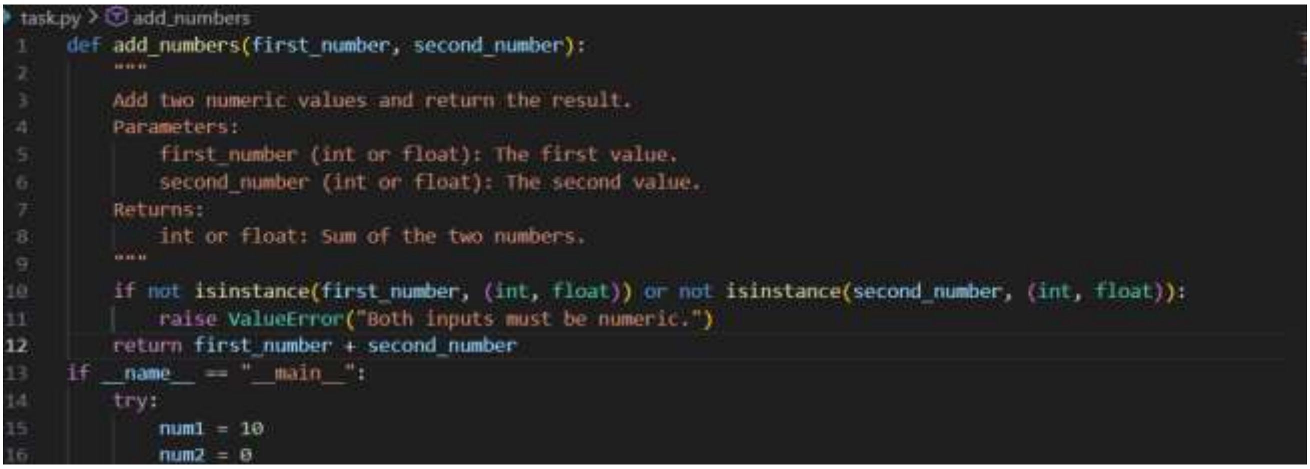
OPTIMIZED CODE



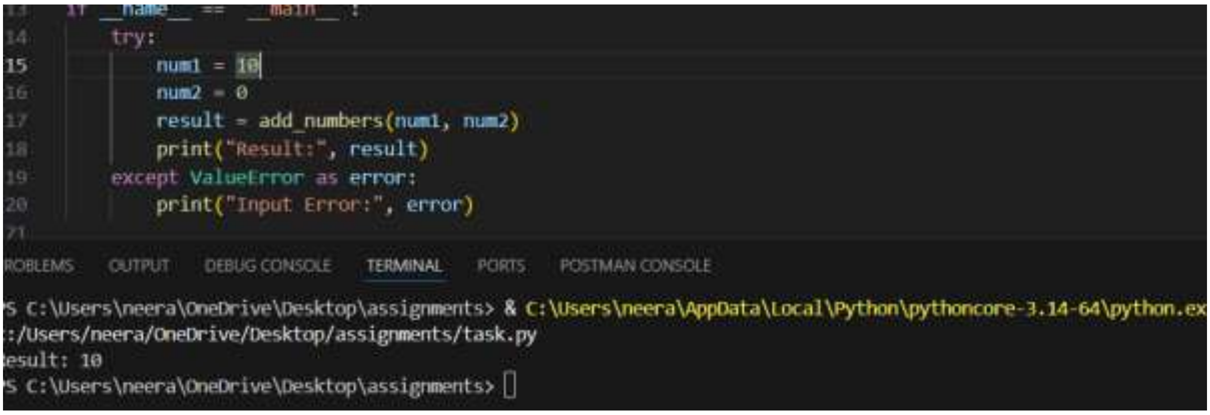
```python
def f(a,b):
 c=a+b
 print(c)

x=10
y=0
f(x,y)

```

```
sqlite3.OperationalError: no such table: users
PS C:\Users\neera\OneDrive\Desktop\assignments>
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:\Users\neera\AppData\Local\Python\pythoncore-3.14-64\python.exe
c:/Users/neera/OneDrive/Desktop/assignments/task.py
10
PS C:\Users\neera\OneDrive\Desktop\assignments> 
```

ORIGINAL CODE:



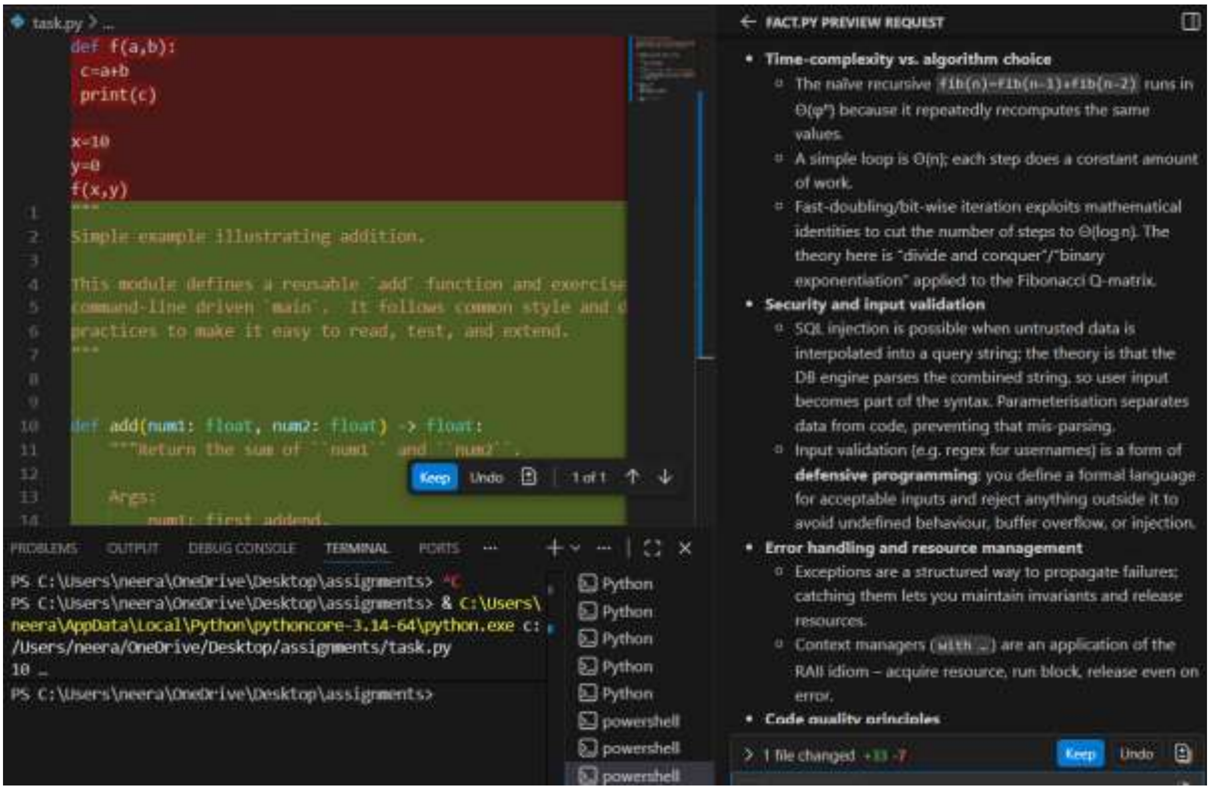
```python
def add_numbers(first_number, second_number):
    """
    Add two numeric values and return the result.
    Parameters:
        first_number (int or float): The first value.
        second_number (int or float): The second value.
    Returns:
        int or float: Sum of the two numbers.
    """
    if not isinstance(first_number, (int, float)) or not isinstance(second_number, (int, float)):
        raise ValueError("Both inputs must be numeric.")
    return first_number + second_number
if __name__ == "__main__":
    try:
        num1 = 10
        num2 = 0
```



```
if __name__ == "__main__":
    try:
        num1 = 10
        num2 = 0
        result = add_numbers(num1, num2)
        print("Result:", result)
    except ValueError as error:
        print("Input Error:", error)
```

```
S C:\Users\neera\OneDrive\Desktop\assignments> & C:\Users\neera\AppData\Local\Python\pythoncore-3.14-64\python.ex
:/Users/neera/OneDrive/Desktop/assignments/task.py
esult: 10
S C:\Users\neera\OneDrive\Desktop\assignments>
```

REVIEW OF AI



```
def f(a,b):
 c=a+b
 print(c)

x=10
y=0
f(x,y)
"""
Simple example illustrating addition.

This module defines a reusable `add` function and exercise
command-line driven `main`.  It follows common style and d
practices to make it easy to read, test, and extend.
"""


def add(num1: float, num2: float) -> float:
    """Return the sum of ``num1`` and ``num2``.

    Args:
        num1: first addend.
```

```
PS C:\Users\neera\OneDrive\Desktop\assignments> ^C
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:\Users\
neera\AppData\Local\Python\pythoncore-3.14-64\python.exe c:
/Users/neera/OneDrive/Desktop/assignments/task.py
10 _
PS C:\Users\neera\OneDrive\Desktop\assignments>
```

FACT.PY PREVIEW REQUEST

- **Time-complexity vs. algorithm choice**
    - The naïve recursive `fib(n)=fib(n-1)+fib(n-2)` runs in Θ(φⁿ) because it repeatedly recomputes the same values.
    - A simple loop is Θ(n); each step does a constant amount of work.
    - Fast-doubling/bit-wise iteration exploits mathematical identities to cut the number of steps to Θ(log n). The theory here is "divide and conquer"/"binary exponentiation" applied to the Fibonacci Q-matrix.
- **Security and input validation**
    - SQL injection is possible when untrusted data is interpolated into a query string; the theory is that the DB engine parses the combined string, so user input becomes part of the syntax. Parameterisation separates data from code, preventing that mis-parsing.
    - Input validation (e.g. regex for usernames) is a form of **defensive programming**: you define a formal language for acceptable inputs and reject anything outside it to avoid undefined behaviour, buffer overflow, or injection.
- **Error handling and resource management**
    - Exceptions are a structured way to propagate failures; catching them lets you maintain invariants and release resources.
    - Context managers (`with …`) are an application of the RAII idiom – acquire resource, run block, release even on error.
- **Code quality principles**

Observation

The AI-based automated code review successfully identified multiple quality issues in the given Python script, including poor readability, unclear naming conventions, inconsistent formatting, lack of documentation, missing error handling, and low maintainability.

The AI tool systematically analyzed the structure, style, and design aspects of the code and highlighted risk areas and code smells before manual review. This demonstrates that AI can effectively act as a preliminary quality assurance tool in the software development lifecycle.

Justification

Using AI for automated code review is justified because it:

- Detects common coding issues early in development
- Ensures adherence to coding standards (e.g., PEP 8)
- Identifies potential risk areas and maintainability concerns
- Reduces human reviewer workload
- Improves overall code consistency across projects
- Helps maintain professional coding practices

AI does not replace human review but enhances efficiency by performing structured preliminary analysis.

Conclusion

AI-assisted automated code review improves code quality by identifying readability issues, naming problems, formatting inconsistencies, missing documentation, lack of error handling, and maintainability risks.

It serves as an effective preliminary review mechanism before manual inspection, ensuring better software quality, consistency, and long-term maintainability.

Thus, AI acts as a smart code reviewer that enhances development standards and supports efficient project management.