2303A51940

Batch:25

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

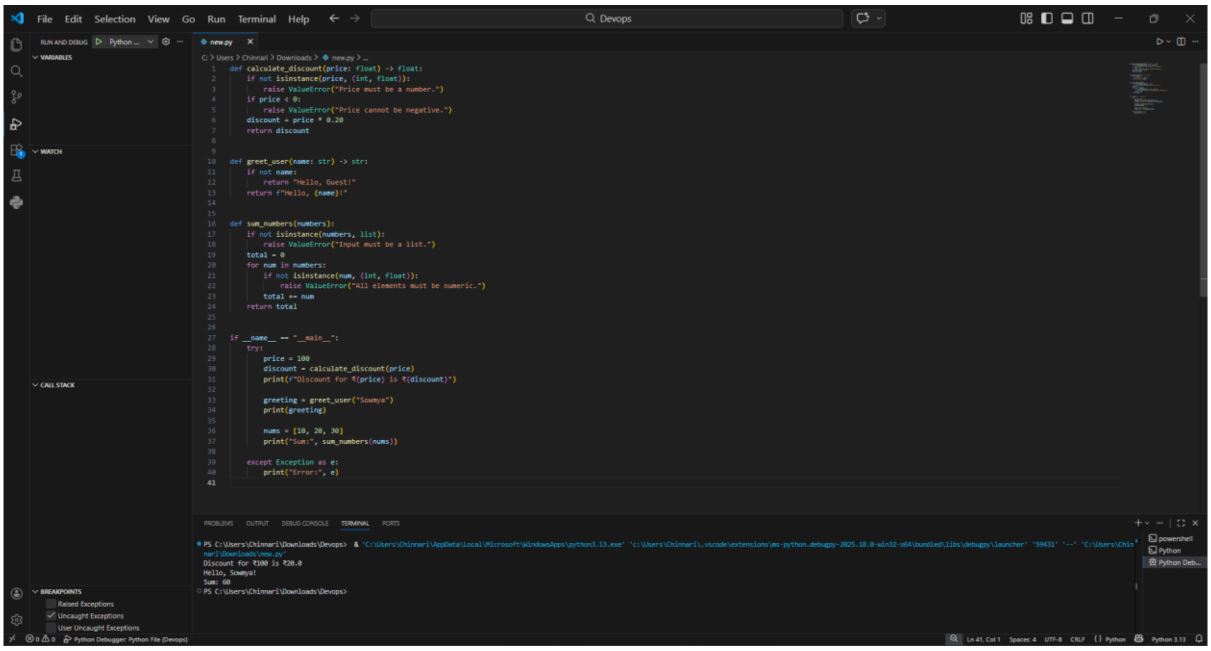
• AI-generated explanation describing:

o Syntax fixes

o Naming corrections

o Structural improvements

• Clean, readable version of the script



```python
def calculate_discount(price: float) -> float:
    if not isinstance(price, (int, float)):
        raise ValueError("Price must be a number.")
    if price < 0:
        raise ValueError("Price cannot be negative.")
    discount = price * 0.20
    return discount


def greet_user(name: str) -> str:
    if not name:
        return "Hello, Guest!"
    return f"Hello, {name}!"


def sum_numbers(numbers):
    if not isinstance(numbers, list):
        raise ValueError("Input must be a list.")
    total = 0
    for num in numbers:
        if not isinstance(num, (int, float)):
            raise ValueError("All elements must be numeric.")
        total += num
    return total


if __name__ == "__main__":
    try:
        price = 100
        discount = calculate_discount(price)
        print(f"Discount for ₹{price} is ₹{discount}")

        greeting = greet_user("Sowmya")
        print(greeting)

        nums = [10, 20, 30]
        print("Sum:", sum_numbers(nums))

    except Exception as e:
        print("Error:", e)
```

```
PS C:\Users\Chinnari\Downloads\Devops>  & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59431' '--' 'C:\Users\Chinnari\Downloads\new.py'
Discount for ₹100 is ₹20.0
Hello, Sowmya!
Sum: 60
PS C:\Users\Chinnari\Downloads\Devops>
```

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:

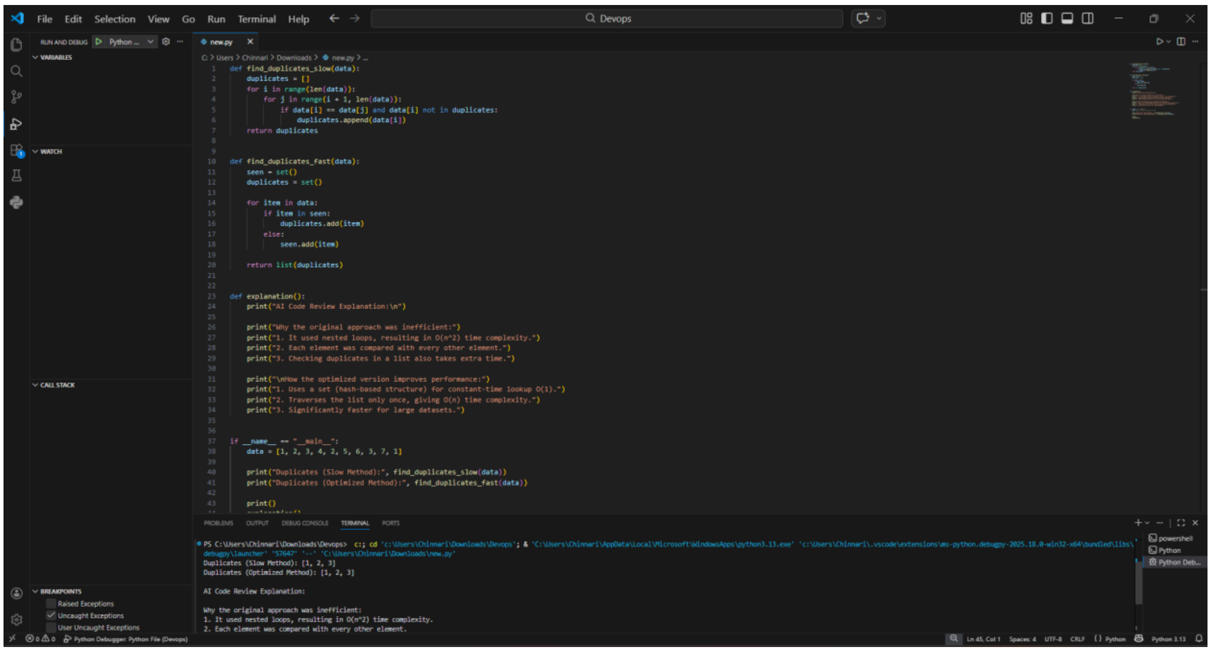
- Analyze the logic for performance bottlenecks
- Refactor the code for better time complexity
- Preserve the correctness of the output

Ask the AI to explain:

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

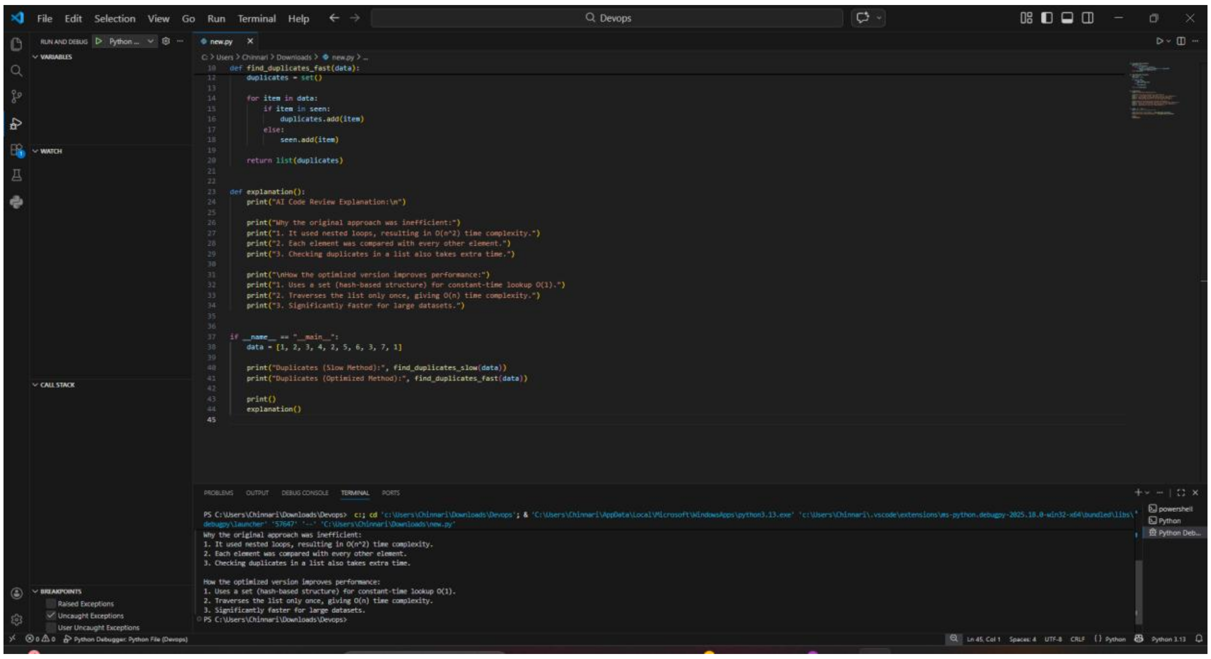
Expected Outcome

- Optimized duplicate-detection logic (e.g., using sets or hash-based structures)
- Improved time complexity
- AI explanation of performance improvement
- Clean, readable implementation



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


def find_duplicates_fast(data):
    seen = set()
    duplicates = set()

    for item in data:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)

    return list(duplicates)


def explanation():
    print("AI Code Review Explanation:\n")

    print("Why the original approach was inefficient:")
    print("1. It used nested loops, resulting in O(n^2) time complexity.")
    print("2. Each element was compared with every other element.")
    print("3. Checking duplicates in a list also takes extra time.")

    print("\nHow the optimized version improves performance:")
    print("1. Uses a set (hash-based structure) for constant-time lookup O(1).")
    print("2. Traverses the list only once, giving O(n) time complexity.")
    print("3. Significantly faster for large datasets.")


if __name__ == "__main__":
    data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 1]

    print("Duplicates (Slow Method):", find_duplicates_slow(data))
    print("Duplicates (Optimized Method):", find_duplicates_fast(data))

    print()
```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57647' '--' 'C:\Users\Chinnari\Downloads\new.py'
Duplicates (Slow Method): [1, 2, 3]
Duplicates (Optimized Method): [1, 2, 3]

AI Code Review Explanation:

Why the original approach was inefficient:
1. It used nested loops, resulting in O(n^2) time complexity.
2. Each element was compared with every other element.
```



```python
def find_duplicates_fast(data):
    duplicates = set()

    for item in data:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)

    return list(duplicates)


def explanation():
    print("AI Code Review Explanation:\n")

    print("Why the original approach was inefficient:")
    print("1. It used nested loops, resulting in O(n^2) time complexity.")
    print("2. Each element was compared with every other element.")
    print("3. Checking duplicates in a list also takes extra time.")

    print("\nHow the optimized version improves performance:")
    print("1. Uses a set (hash-based structure) for constant-time lookup O(1).")
    print("2. Traverses the list only once, giving O(n) time complexity.")
    print("3. Significantly faster for large datasets.")


if __name__ == "__main__":
    data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 1]

    print("Duplicates (Slow Method):", find_duplicates_slow(data))
    print("Duplicates (Optimized Method):", find_duplicates_fast(data))

    print()
    explanation()
```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57647' '--' 'C:\Users\Chinnari\Downloads\new.py'
Why the original approach was inefficient:
1. It used nested loops, resulting in O(n^2) time complexity.
2. Each element was compared with every other element.
3. Checking duplicates in a list also takes extra time.

How the optimized version improves performance:
1. Uses a set (hash-based structure) for constant-time lookup O(1).
2. Traverses the list only once, giving O(n) time complexity.
3. Significantly faster for large datasets.
PS C:\Users\Chinnari\Downloads\Devops>
```

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

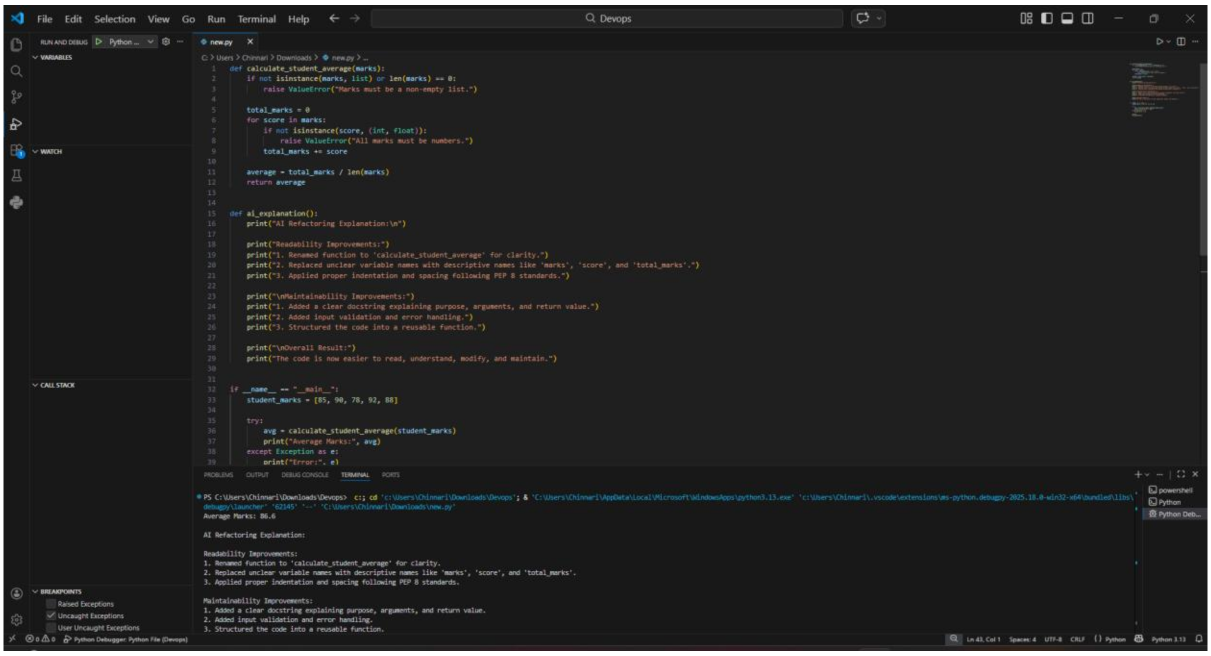
• Clean, well-structured code

• Descriptive function and variable names

• Proper indentation and formatting

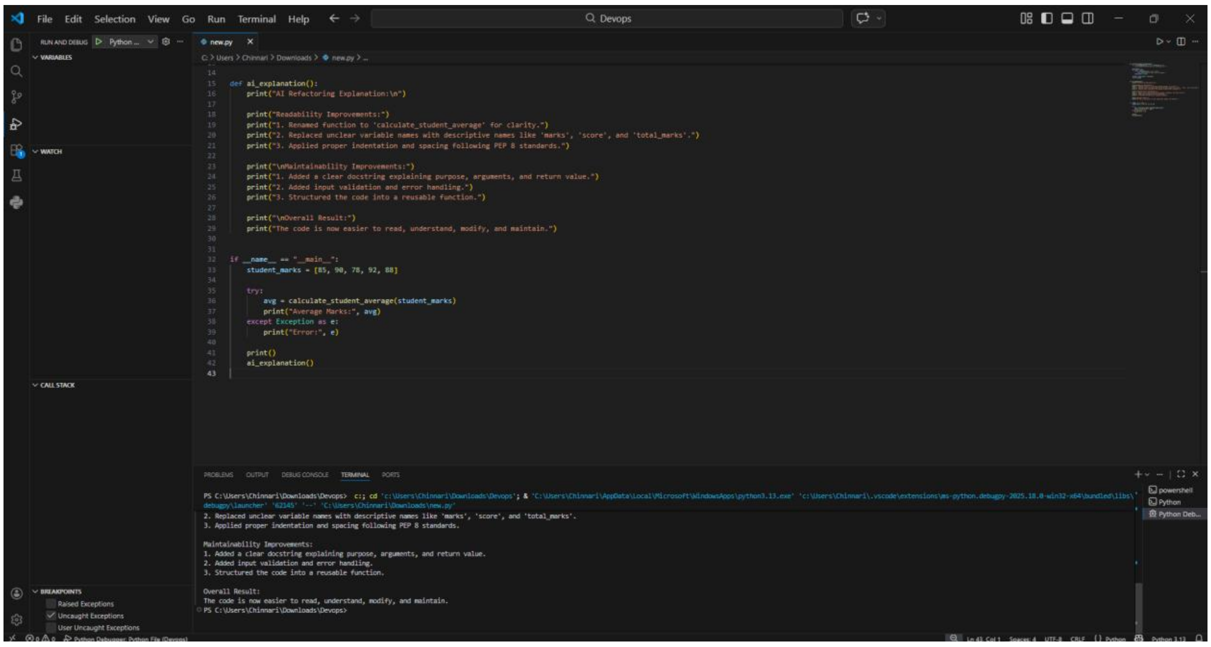
• Docstrings explaining the function purpose

• AI explanation of readability improvements



```python
def calculate_student_average(marks):
    if not isinstance(marks, list) or len(marks) == 0:
        raise ValueError("Marks must be a non-empty list.")

    total_marks = 0
    for score in marks:
        if not isinstance(score, (int, float)):
            raise ValueError("All marks must be numbers.")
        total_marks += score

    average = total_marks / len(marks)
    return average


def ai_explanation():
    print("AI Refactoring Explanation:\n")

    print("Readability Improvements:")
    print("1. Renamed function to 'calculate_student_average' for clarity.")
    print("2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.")
    print("3. Applied proper indentation and spacing following PEP 8 standards.")

    print("\nMaintainability Improvements:")
    print("1. Added a clear docstring explaining purpose, arguments, and return value.")
    print("2. Added input validation and error handling.")
    print("3. Structured the code into a reusable function.")

    print("\nOverall Result:")
    print("The code is now easier to read, understand, modify, and maintain.")


if __name__ == "__main__":
    student_marks = [85, 90, 78, 92, 88]

    try:
        avg = calculate_student_average(student_marks)
        print("Average Marks:", avg)
    except Exception as e:
        print("Error:", e)
```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62145' '--' 'C:\Users\Chinnari\Downloads\new.py'
Average Marks: 86.6

AI Refactoring Explanation:

Readability Improvements:
1. Renamed function to 'calculate_student_average' for clarity.
2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.
3. Applied proper indentation and spacing following PEP 8 standards.

Maintainability Improvements:
1. Added a clear docstring explaining purpose, arguments, and return value.
2. Added input validation and error handling.
3. Structured the code into a reusable function.
```


File Edit Selection View Go Run Terminal Help
Devops
RUN AND DEBUG
VARIABLES
WATCH
CALL STACK
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
User Uncaught Exceptions
new.py
C: > Users > Chinnari > Downloads > new.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ln 43, Col 1 Spaces: 4 UTF-8 CRLF Python

```python

def ai_explanation():
    print("AI Refactoring Explanation:\n")

    print("Readability Improvements:")
    print("1. Renamed function to 'calculate_student_average' for clarity.")
    print("2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.")
    print("3. Applied proper indentation and spacing following PEP 8 standards.")

    print("\nMaintainability Improvements:")
    print("1. Added a clear docstring explaining purpose, arguments, and return value.")
    print("2. Added input validation and error handling.")
    print("3. Structured the code into a reusable function.")

    print("\nOverall Result:")
    print("The code is now easier to read, understand, modify, and maintain.")


if __name__ == "__main__":
    student_marks = [85, 90, 78, 92, 88]

    try:
        avg = calculate_student_average(student_marks)
        print("Average Marks:", avg)
    except Exception as e:
        print("Error:", e)

    print()
    ai_explanation()

```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62145' '--' 'C:\Users\Chinnari\Downloads\new.py'
2. Replaced unclear variable names with descriptive names like 'marks', 'score', and 'total_marks'.
3. Applied proper indentation and spacing following PEP 8 standards.

Maintainability Improvements:
1. Added a clear docstring explaining purpose, arguments, and return value.
2. Added input validation and error handling.
3. Structured the code into a reusable function.

Overall Result:
The code is now easier to read, understand, modify, and maintain.
PS C:\Users\Chinnari\Downloads\Devops>
```

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

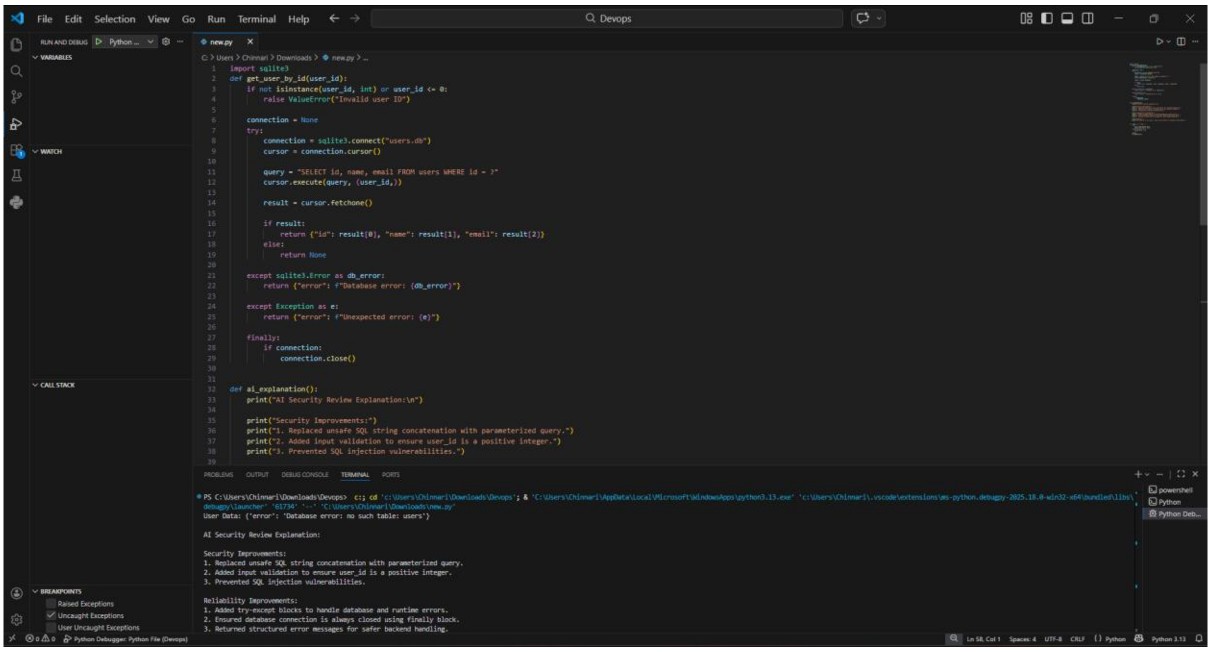
• Secure SQL queries using parameterized statements

• Input validation logic

• Try-except blocks for runtime safety

• AI-generated explanation of security improvements

• Production-ready code structure give code for this remove comments



```python
import sqlite3
def get_user_by_id(user_id):
    if not isinstance(user_id, int) or user_id <= 0:
        raise ValueError("Invalid user ID")

    connection = None
    try:
        connection = sqlite3.connect("users.db")
        cursor = connection.cursor()

        query = "SELECT id, name, email FROM users WHERE id = ?"
        cursor.execute(query, (user_id,))

        result = cursor.fetchone()

        if result:
            return {"id": result[0], "name": result[1], "email": result[2]}
        else:
            return None

    except sqlite3.Error as db_error:
        return {"error": f"Database error: {db_error}"}

    except Exception as e:
        return {"error": f"Unexpected error: {e}"}

    finally:
        if connection:
            connection.close()


def ai_explanation():
    print("AI Security Review Explanation:\n")

    print("Security Improvements:")
    print("1. Replaced unsafe SQL string concatenation with parameterized query.")
    print("2. Added input validation to ensure user_id is a positive integer.")
    print("3. Prevented SQL injection vulnerabilities.")
```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61734' '--' 'C:\Users\Chinnari\Downloads\new.py'
User Data: {'error': 'Database error: no such table: users'}

AI Security Review Explanation:

Security Improvements:
1. Replaced unsafe SQL string concatenation with parameterized query.
2. Added input validation to ensure user_id is a positive integer.
3. Prevented SQL injection vulnerabilities.

Reliability Improvements:
1. Added try-except blocks to handle database and runtime errors.
2. Ensured database connection is always closed using finally block.
3. Returned structured error messages for safer backend handling.
```

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

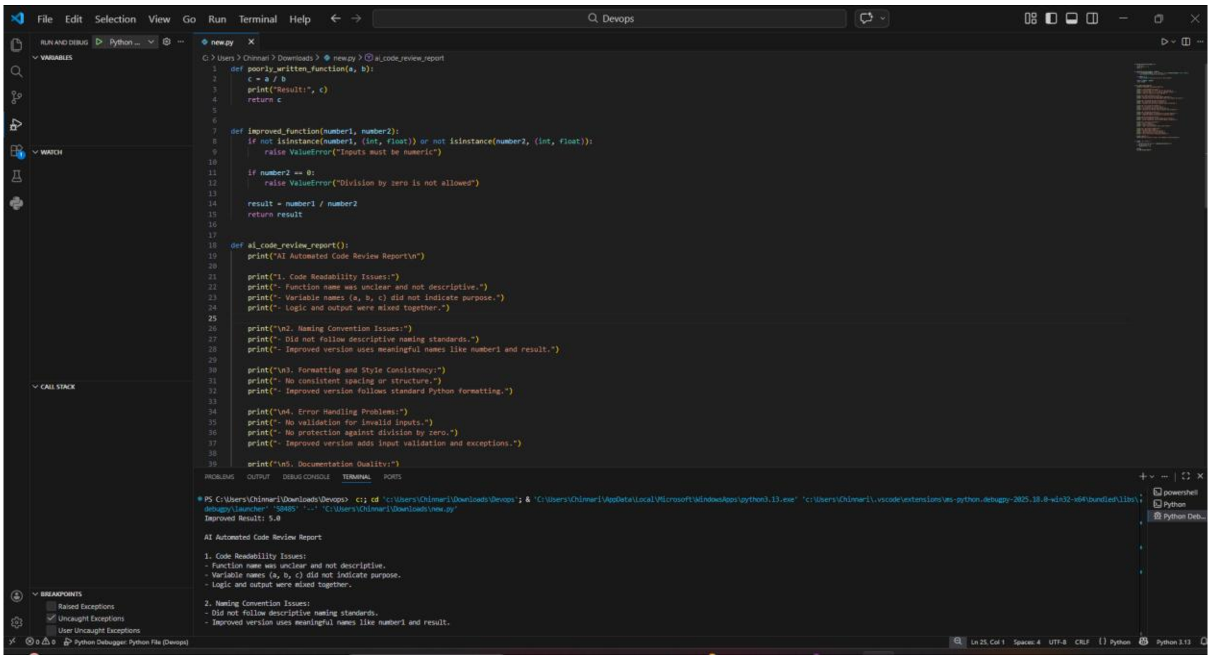
o Code smell detection

o Improvement suggestions

• Optional improved version of the code

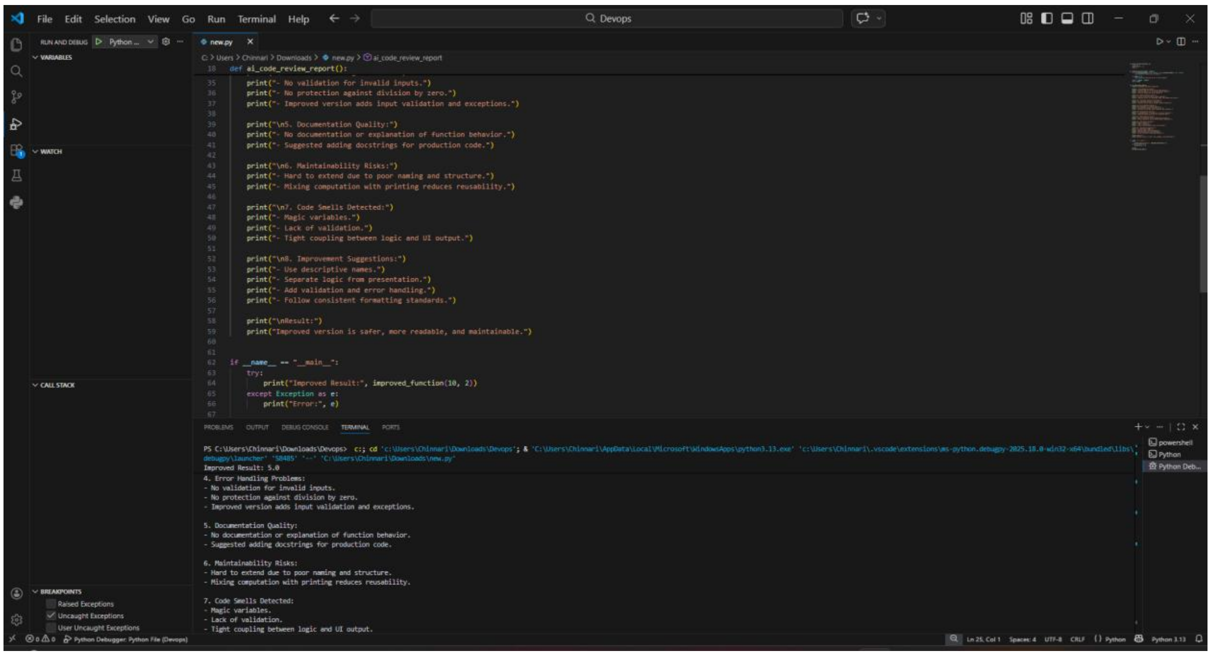
• Demonstration of AI as a code reviewer, not just a code

Generator



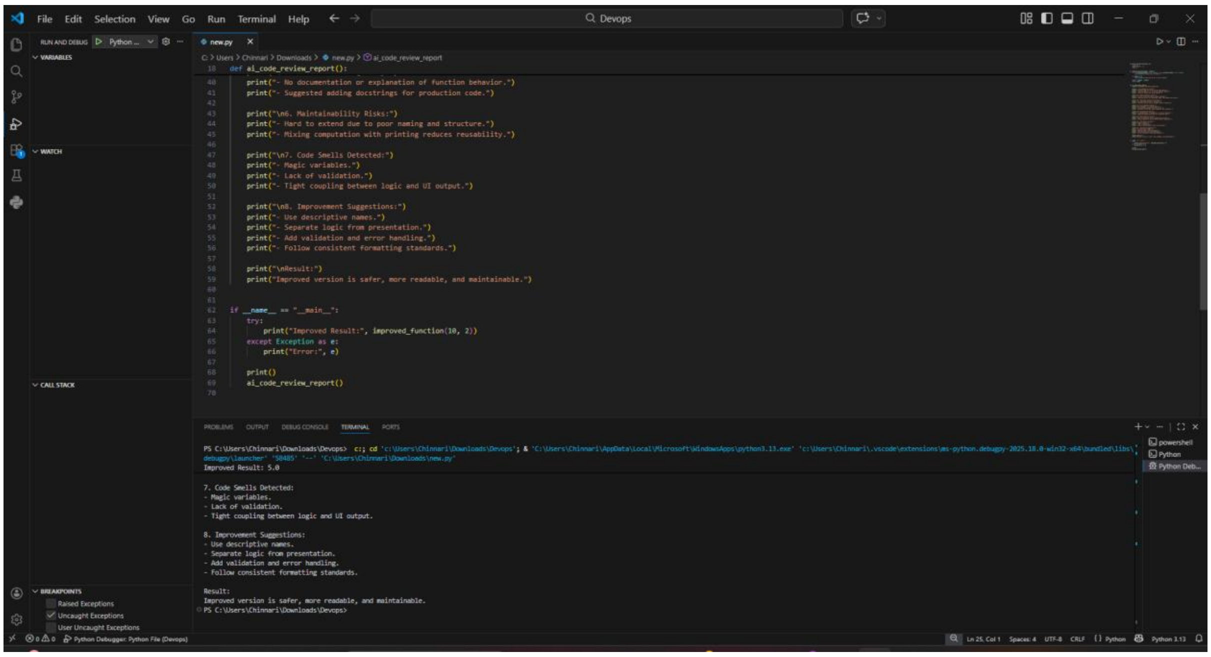
```python
def poorly_written_function(a, b):
    c = a / b
    print("Result:", c)
    return c


def improved_function(number1, number2):
    if not isinstance(number1, (int, float)) or not isinstance(number2, (int, float)):
        raise ValueError("Inputs must be numeric")

    if number2 == 0:
        raise ValueError("Division by zero is not allowed")

    result = number1 / number2
    return result


def ai_code_review_report():
    print("AI Automated Code Review Report\n")

    print("1. Code Readability Issues:")
    print("- Function name was unclear and not descriptive.")
    print("- Variable names (a, b, c) did not indicate purpose.")
    print("- Logic and output were mixed together.")

    print("\n2. Naming Convention Issues:")
    print("- Did not follow descriptive naming standards.")
    print("- Improved version uses meaningful names like number1 and result.")

    print("\n3. Formatting and Style Consistency:")
    print("- No consistent spacing or structure.")
    print("- Improved version follows standard Python formatting.")

    print("\n4. Error Handling Problems:")
    print("- No validation for invalid inputs.")
    print("- No protection against division by zero.")
    print("- Improved version adds input validation and exceptions.")

    print("\n5. Documentation Quality:")
```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58485' '--' 'C:\Users\Chinnari\Downloads\new.py'
Improved Result: 5.0

AI Automated Code Review Report

1. Code Readability Issues:
- Function name was unclear and not descriptive.
- Variable names (a, b, c) did not indicate purpose.
- Logic and output were mixed together.

2. Naming Convention Issues:
- Did not follow descriptive naming standards.
- Improved version uses meaningful names like number1 and result.
```



```python
def ai_code_review_report():
    print("- No validation for invalid inputs.")
    print("- No protection against division by zero.")
    print("- Improved version adds input validation and exceptions.")

    print("\n5. Documentation Quality:")
    print("- No documentation or explanation of function behavior.")
    print("- Suggested adding docstrings for production code.")

    print("\n6. Maintainability Risks:")
    print("- Hard to extend due to poor naming and structure.")
    print("- Mixing computation with printing reduces reusability.")

    print("\n7. Code Smells Detected:")
    print("- Magic variables.")
    print("- Lack of validation.")
    print("- Tight coupling between logic and UI output.")

    print("\n8. Improvement Suggestions:")
    print("- Use descriptive names.")
    print("- Separate logic from presentation.")
    print("- Add validation and error handling.")
    print("- Follow consistent formatting standards.")

    print("\nResult:")
    print("Improved version is safer, more readable, and maintainable.")


if __name__ == "__main__":
    try:
        print("Improved Result:", improved_function(10, 2))
    except Exception as e:
        print("Error:", e)
```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58485' '--' 'C:\Users\Chinnari\Downloads\new.py'
Improved Result: 5.0
4. Error Handling Problems:
- No validation for invalid inputs.
- No protection against division by zero.
- Improved version adds input validation and exceptions.

5. Documentation Quality:
- No documentation or explanation of function behavior.
- Suggested adding docstrings for production code.

6. Maintainability Risks:
- Hard to extend due to poor naming and structure.
- Mixing computation with printing reduces reusability.

7. Code Smells Detected:
- Magic variables.
- Lack of validation.
- Tight coupling between logic and UI output.
```



```python
def ai_code_review_report():
    print("- No documentation or explanation of function behavior.")
    print("- Suggested adding docstrings for production code.")

    print("\n6. Maintainability Risks:")
    print("- Hard to extend due to poor naming and structure.")
    print("- Mixing computation with printing reduces reusability.")

    print("\n7. Code Smells Detected:")
    print("- Magic variables.")
    print("- Lack of validation.")
    print("- Tight coupling between logic and UI output.")

    print("\n8. Improvement Suggestions:")
    print("- Use descriptive names.")
    print("- Separate logic from presentation.")
    print("- Add validation and error handling.")
    print("- Follow consistent formatting standards.")

    print("\nResult:")
    print("Improved version is safer, more readable, and maintainable.")


if __name__ == "__main__":
    try:
        print("Improved Result:", improved_function(10, 2))
    except Exception as e:
        print("Error:", e)

    print()
    ai_code_review_report()
```

```
PS C:\Users\Chinnari\Downloads\Devops>  c:; cd 'c:\Users\Chinnari\Downloads\Devops'; & 'C:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58485' '--' 'C:\Users\Chinnari\Downloads\new.py'
Improved Result: 5.0

7. Code Smells Detected:
- Magic variables.
- Lack of validation.
- Tight coupling between logic and UI output.

8. Improvement Suggestions:
- Use descriptive names.
- Separate logic from presentation.
- Add validation and error handling.
- Follow consistent formatting standards.

Result:
Improved version is safer, more readable, and maintainable.
PS C:\Users\Chinnari\Downloads\Devops>
```