

Name:P Pranay Kumar H.No:2303A51829

Batch:26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name:B. Tech	Assignment Type: Lab		Academic Year:2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s)Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju Mr. G. Kranthi		
Course Code	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/I	Regulation	R23
Date and Day of Assignment	Week 5 - Thursday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All Batches
AssignmentNumber: 10.4 (Present assignment number)/ 24 (Total number of assignments)			
Q.No.	Question		Expected Time to complete

1	<p>Lab 9 – Code Review and Quality: Using AI to Improve Code Quality and Readability</p> <p>Lab Objectives</p> <ul style="list-style-type: none"> • 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 	Week 5
	<p>Task 1: AI-Assisted Syntax and Code Quality Review</p> <p>Scenario</p> <p>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.</p> <p>Task Description</p> <p>You are given a Python script containing:</p> <ul style="list-style-type: none"> • Syntax errors • Indentation issues • Incorrect variable names • Faulty function calls <p>Use an AI tool (GitHub Copilot / Cursor AI) to:</p> <ul style="list-style-type: none"> • Identify all syntactic and structural errors • Correct them systematically • Generate an explanation of each fix made <p>Expected Outcome</p> <ul style="list-style-type: none"> • Fully corrected and executable Python code • AI-generated explanation describing: <ul style="list-style-type: none"> ◦ Syntax fixes ◦ Naming corrections ◦ Structural improvements • Clean, readable version of the script 	

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The main area displays a Python script named `AAC A 10.4.py`. The code contains a function `calculate_average` and a `main` function. The `main` function prints the average of a list of numbers. The AI-assisted troubleshooting panel on the right shows that the script was updated to fix errors related to missing colons and concatenating floats with strings. The terminal below shows the command to run the script and its output, which is `The average is: 3.0`.

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

```

1 def find_duplicates_efficient(lst):
2     seen = set()
3     duplicates = set()
4     for item in lst:
5         if item in seen:
6             duplicates.add(item)
7         else:
8             seen.add(item)
9     return list(duplicates)
10
11 if __name__ == "__main__":
12     print(find_duplicates_efficient([1,2,3,2,4,1]))

```

The optimized duplicate-detection function uses a set-based approach for $O(n)$ time complexity instead of the original $O(n^2)$ nested loops, which repeatedly check each element against all others—causing exponential slowdown on large datasets. This version performs a single pass, tracking seen items and collecting duplicates efficiently while maintaining correct output.

```

def find_duplicates_efficient(lst):
    seen = set()
    duplicates = set()
    for item in lst:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)
    return list(duplicates)

```

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

```
C:\> Users > bash > Downloads > AAC A 10.4.py > find_duplicate_values
1 def find_duplicate_values(input_list):
2
3     seen = set()
4     duplicates = set()
5     for item in input_list:
6         if item in seen:
7             duplicates.add(item)
8         else:
9             seen.add(item)
10    return list(duplicates)
11
12 if __name__ == "__main__":
13     test_list = [1, 2, 3, 2, 4, 1]
14     result = find_duplicate_values(test_list)
15     print(result)

The average is: 3.0
```

The code has been refactored for better readability and maintainability: the function name is now descriptive ("find_duplicate_values" instead of "find_duplicates_efficient"), variable names are clearer ("input_list" vs. "lst"), a docstring documents the purpose and parameters, and the main block uses meaningful variable names with proper formatting. This follows PEP 8 standards, making the code easier to understand and maintain without altering functionality.

The code has been refactored for better readability and maintainability: the function name is now descriptive ("find_duplicate_values" instead of "find_duplicates_efficient"), variable names are clearer ("input_list" vs. "lst"), a docstring documents the purpose and parameters, and the main block uses meaningful variable names with proper formatting. This follows PEP 8 standards, making the code easier to understand and maintain without altering functionality.

The code has been refactored for better readability and maintainability: the function name is now descriptive ("find_duplicate_values" instead of "find_duplicates_efficient"), variable names are clearer ("input_list" vs. "lst"), a docstring documents the purpose and parameters, and the main block uses meaningful variable names with proper formatting. This follows PEP 8 standards, making the code easier to understand and maintain without altering functionality.

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

The screenshot shows a code editor interface with two main panes. The left pane displays the original Python script 'AAC A 10.4.py' containing unsafe code. The right pane shows the AI-generated refactored code, which includes parameterized SQL queries, input validation, and try-except blocks. Below the code editor is a terminal window showing command-line interactions and a status bar indicating 'Grok Code Fast 1 / 0.25x'.

```
import sqlite
def get_user_data(user_id):
    if not isinstance(user_id, int) or user_id <= 0:
        raise ValueError("User ID must be a positive integer")
    try:
        conn = sqlite.connect('database.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
        result = cursor.fetchone()
        return result
    except sqlite3.Error as e:
        print(f"Database error: {e}")
        return None
    finally:
        if 'conn' in locals():
            conn.close()

if __name__ == "__main__":
    try:
        data = get_user_data(1)
        print(data)
    except ValueError as e:
        print(e)
```

```
import sqlite3
from sqlite3 import Error

def get_user_data(user_id):
    if not isinstance(user_id, int) or user_id <= 0:
        raise ValueError("User ID must be a positive integer")
    try:
        conn = sqlite3.connect('database.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
        result = cursor.fetchone()
        return result
    except Error as e:
        print(f"Database error: {e}")
        return None
    finally:
        if 'conn' in locals():
            conn.close()
```

The screenshot shows a code editor interface with several windows:

- Editor Window:** Displays the Python script `AAC A 10.4.py`. The code includes a function `get_user_data` that uses a cursor to select user data from a database. It handles a `sqlite3.Error` exception by printing the error message and returning `None`. It also includes a `finally` block to close the connection if it was opened.
- Terminal Window:** Shows the command `python "c:\Users\shash\Downloads\AAC A 10.4.py"` being run in a Windows terminal. The output indicates that no database file exists, so no user data is printed.
- CHAT Window:** Shows an AI-generated snippet for error handling. It prints the database error if one occurs, returns `None`, and closes the connection if it was opened.
- Bottom Status Bar:** Shows the status bar with various icons and the text "Grok Code Fast 1 • 0.25x".

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:
 - Code readability
 - Naming conventions
 - Formatting and style consistency
 - Error handling
 - Documentation quality
 - 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:
 - Identified quality issues
 - Risk areas
 - Code smell detection
 - Improvement suggestions
- Optional improved version of the code
- Demonstration of AI as a **code reviewer**, not just a code generator

```

Welcome AAC A 10.4.py ...
C:\> Users > shash > Downloads > AAC A 10.4.py ...
1 import sqlite3
2
3 def getdata(id):
4     try:
5         conn = sqlite3.connect('db')
6         cur = conn.cursor()
7         cur.execute("SELECT * FROM users WHERE id = " + str(id))
8         res = cur.fetchone()
9         return res
10    except sqlite3.Error as e:
11        print("Database error: (e)")
12        return None
13    finally:
14        if 'conn' in locals():
15            conn.close()
16
17 if __name__ == "__main__":
18     print(getdata(1))
19

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\shash\Downloads> c; cd 'c:\Users\shash\Downloads'; & 'c:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\Launcher' '5055' '-t' 'c:\Users\shash\Downloads\AAC A 10.4.py'
Database error: no such table: users
None

PS C:\Users\shash\Downloads> c; cd 'c:\Users\shash\Downloads'; & 'c:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\Launcher' '61177' '-t' 'c:\Users\shash\Downloads\AAC A 10.4.py'
Database error: no such table: users
None

PS C:\Users\shash\Downloads> []

2 Hidden T...

CHAT ERROR TROUBLESHOOTING ASSISTANCE NEEDED

Edited AAC A 10.4.py

python "c:\Users\shash\Downloads\AAC A 10.4.py", 1:
print(getdata(1))
~~~~~^MAA

File "c:\Users\shash\Downloads\AAC A 10.4.py", 1:
cur.execute("SELECT \* FROM users WHERE id = ").
~~~~~^MAA

sqlite3.OperationalError: no such table: users

The print statement has been added. Running the script results in a database error (no 'db' file or 'users' table exists), causing a crash as expected from the poorly written code with no error handling. The output is the traceback shown above.

fix error

✓ Reviewed and updated AAC A 10.4.py

✓ Edited AAC A 10.4.py +12 -6

python "c:\Users\shash\Downloads\AAC A 10.4.py"

The error has been fixed by adding try-except-finally blocks for proper error handling. The script now catches the database error, prints a

Grok Code Fast 1

In 19, Col 1 Spaces: 4 UTE: 8 Python 3.12.9 (Shashidhar) Go Live

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots