

CH VASU

2303A51618

BATCH 25

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	23CS002 PC304	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)			
	Question	Expected Time to complete	
1	<p><b>Lab 9 – Code Review and Quality: Using AI to Improve Code Quality and Readability</b></p> <p><b>Lab Objectives</b></p> <ul style="list-style-type: none"> <li>• Use AI for automated code review and quality enhancement.</li> <li>• Identify and fix syntax, logical, performance, and security issues in Python code.</li> <li>• Improve readability and maintainability through structured refactoring and comments.</li> <li>• Apply prompt engineering for targeted improvements.</li> <li>• Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices</li> </ul>	Week 5	
	<p><b>Task 1: AI-Assisted Syntax and Code Quality Review</b></p> <p><b>Scenario</b> 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><b>Task Description</b> You are given a Python script containing:</p> <ul style="list-style-type: none"> <li>• Syntax errors</li> <li>• Indentation issues</li> <li>• Incorrect variable names</li> <li>• Faulty function calls</li> </ul> <p>Use an AI tool (GitHub Copilot / Cursor AI) to:</p>		

- 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:
  - Syntax fixes
  - Naming corrections
  - Structural improvements
- Clean, readable version of the script

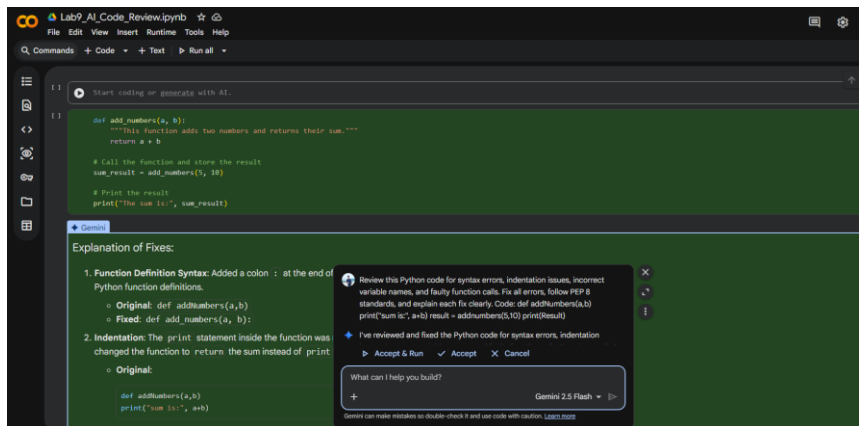
Sample :

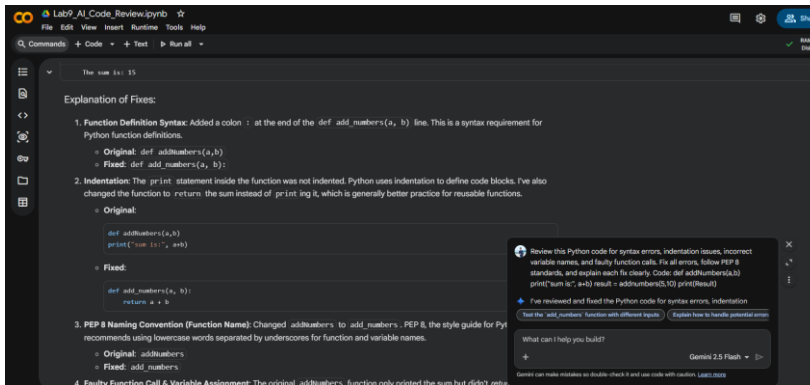
Review this Python code for syntax errors, indentation issues, incorrect variable names, and faulty function calls. Fix all errors, follow PEP 8 standards, and explain each fix clearly.

Code:

```
def addNumbers(a,b)
print("sum is:", a+b)
```

```
result = addnumbers(5,10)
print(Result)
```





I've reviewed and fixed the Python code for syntax errors, indentation issues, incorrect variable names, and faulty function calls. I've also applied PEP 8 standards. Here's the corrected code with explanations for each fix: The code in cell dbf790b3 executed successfully, and the output **The sum is: 15** confirms that the `add_numbers` function correctly calculated the sum of 5 and 10, and the result was printed as expected.

## 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**

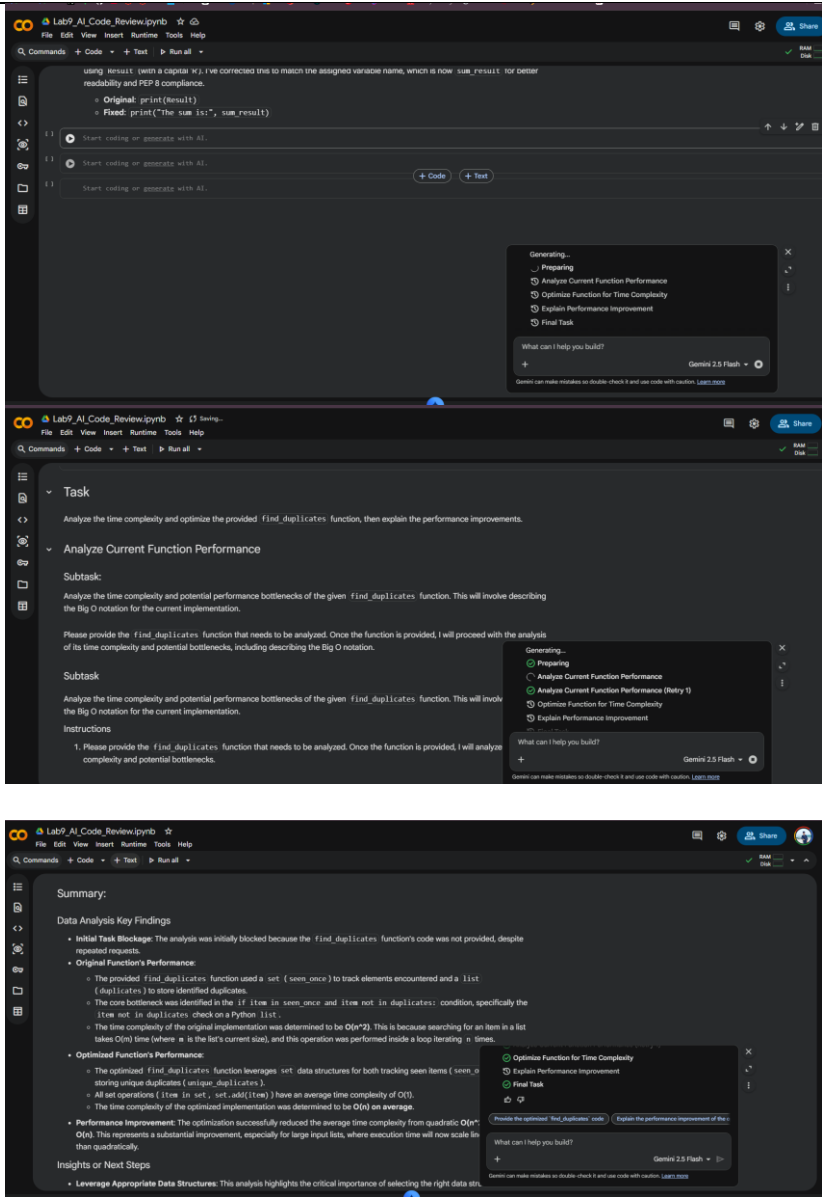
- Optimized duplicate-detection logic (e.g., using sets or hash-based structures)
- Improved time complexity
- AI explanation of performance improvement
- Clean, readable implementation

Sample:

Analyze this Python function for performance issues. Optimize it for better time complexity using efficient data structures and explain why the optimized version is faster.

Code:

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

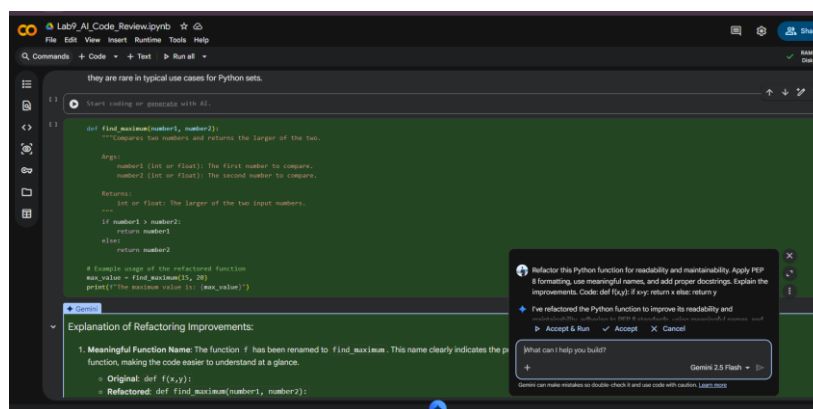
## Expected Outcome

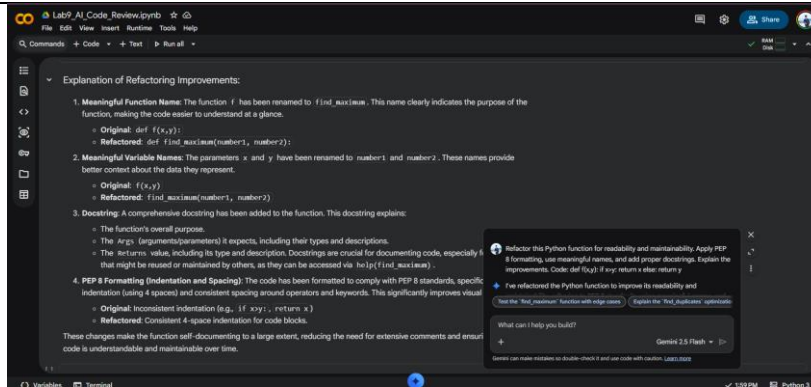
- Clean, well-structured code
- Descriptive function and variable names
- Proper indentation and formatting
- Docstrings explaining the function purpose
- AI explanation of readability improvements

Refactor this Python function for readability and maintainability. Apply PEP 8 formatting, use meaningful names, and add proper docstrings. Explain the improvements.

Code:

```
def f(x,y):
    if x>y:
        return x
    else:
        return y
```





I've refactored the Python function to improve its readability and maintainability, adhering to PEP 8 standards, using meaningful names, and adding a clear docstring. Below is the refactored code and an explanation of the improvements.

The refactored `find_maximum` function executed successfully. The output "The maximum value is: 20" correctly shows that the function identified 20 as the larger value between 15 and 20, as intended by the refactoring.

## 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



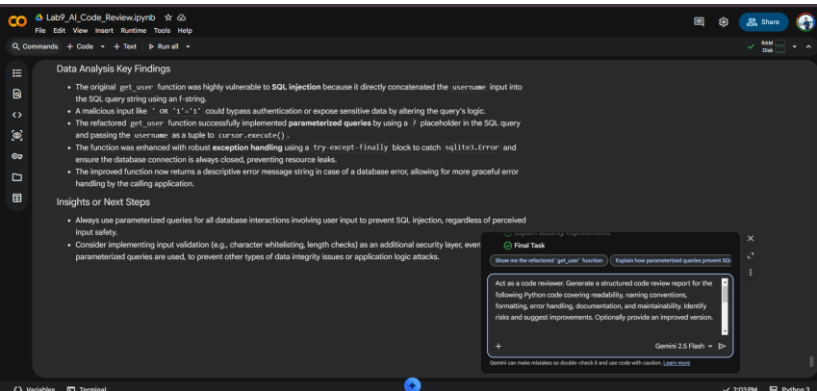
- Production-ready code structure

Review this Python code for security vulnerabilities. Refactor it using safe SQL practices, input validation, and proper exception handling. Explain each security improvement.

Code:

```
import sqlite3

def get_user(username):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE name = '" +
username + "'"
    cursor.execute(query)
    return cursor.fetchone()
```



## 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**.

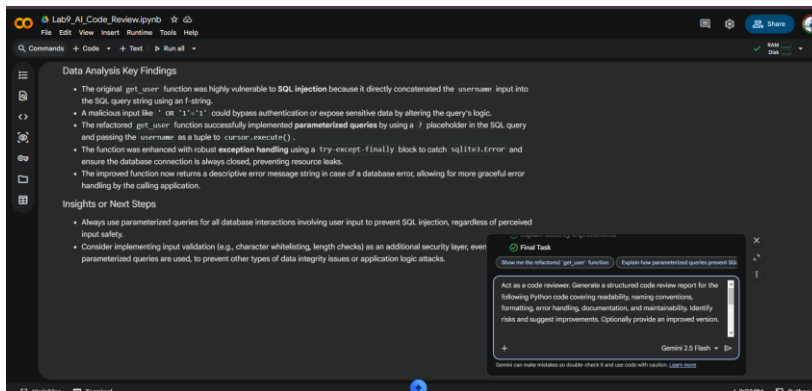
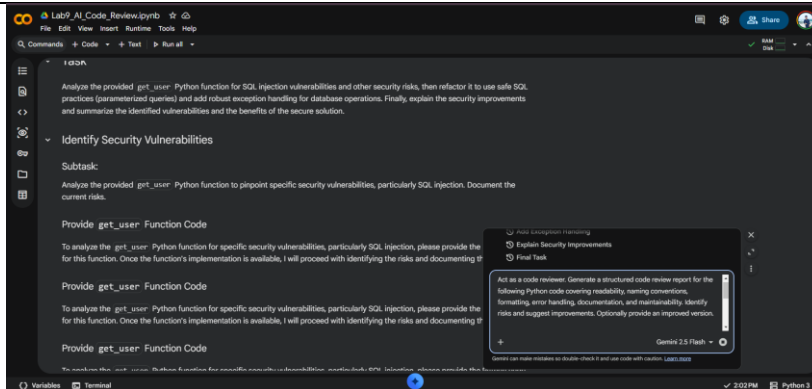
### **Expected Outcome**

- 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

**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**

Act as a code reviewer. Generate a structured code review report for the following Python code covering readability, naming conventions, formatting, error handling, documentation, and maintainability. Identify risks and suggest improvements. Optionally provide an improved version.

```
Code:
def calc(a,b,c):
    r=a+b
    if c==1:
        return r
    else:
        return a*b
```



## nsights or Next Steps

Always use parameterized queries for all database interactions involving user input to prevent SQL injection, regardless of perceived input safety.

Consider implementing input validation (e.g., character whitelisting, length checks) as an additional security layer, even when parameterized queries are used, to prevent other types of data integrity issues or application logic attacks.

[Colab paid products](#) - [Cancel contracts here](#)



Review this Python code for syntax errors, indentation issues, incorrect variable names, and faulty function calls. Fix all errors, follow PEP 8 standards, and explain each fix clearly. Code: 

```
def addNumbers(a,b)
print("sum is:", a+b) result = addnumbers(5,10) print(Result)
spark
```

I've reviewed and fixed the Python code for syntax errors, indentation issues, incorrect variable names, and faulty function calls. I've also applied PEP 8 standards. Here's the corrected code with explanations for each fix:

The code in cell dbf790b3 executed successfully, and the output `The sum is: 15` confirms that the `add_numbers` function correctly calculated the sum of 5 and 10, and the result was printed as expected.

thumb\_up

thumb\_down




Analyze this Python function for performance issues. Optimize it for better time complexity using efficient data structures and explain why the optimized version is faster. Code: 

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

Certainly! I've created a plan to analyze the provided Python function for performance issues, optimize it using efficient data structures, and explain the improvements.

**Analyze Current Function Performance:** Analyze the time complexity and potential performance bottlenecks of the given `find_duplicates` function. This will involve describing the Big O notation for the current implementation.

**Optimize Function for Time Complexity:** Develop an optimized version of the `find_duplicates` function using more efficient data

<p>structures (e.g., sets) to improve its time complexity. The optimized code will be provided.</p> <p><b>Explain Performance Improvement:</b> Clearly explain why the optimized version of the function is faster, comparing the time complexities of the original and optimized implementations. Detail how the choice of data structures contributes to the performance gain.</p> <p><b>Final Task:</b> Summarize the analysis, the optimized solution, and the key reasons for its improved performance.</p> <p>Generating...</p> <p>check_circle</p> <p>Preparing</p> <p>check_circle</p> <p>Analyze Current Function Performance</p> <p>check_circle</p> <p>Analyze Current Function Performance (Retry 1)</p> <p>check_circle</p> <p>Optimize Function for Time Complexity</p> <p>history</p> <p>Explain Performance Improvement</p> <p>check_circle</p> <p>Final Task</p> <p>thumb_up</p> <p>thumb_down</p>  <p>Refactor this Python function for readability and maintainability. Apply PEP 8 formatting, use meaningful names, and add proper docstrings. Explain the improvements. Code: <code>def f(x,y): if x&gt;y: return x else: return y</code></p> <p>spark</p> <p>I've refactored the Python function to improve its readability and maintainability, adhering to PEP 8 standards, using meaningful names, and adding a clear docstring. Below is the refactored code and an explanation of the improvements.</p> <p>The refactored <code>find_maximum</code> function executed successfully. The output <code>The maximum value is: 20</code> correctly shows that the function identified 20 as the larger value between 15 and 20, as intended by the refactoring.</p>	
--	--

thumb\_up

thumb\_down



Review this Python code for security vulnerabilities. Refactor it using safe SQL practices, input validation, and proper exception handling.

Explain each security improvement. Code: 

```
import sqlite3
def get_user(username):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE name = " + username + ""
    cursor.execute(query)
    return cursor.fetchone()
```

spark

I've reviewed your Python code and identified potential security vulnerabilities. I've generated a plan to refactor it using safe SQL practices, input validation, and proper exception handling, along with explanations for each security improvement. Please review the plan.

**Identify Security Vulnerabilities:** Analyze the provided `get_user` Python function to pinpoint specific security vulnerabilities, particularly SQL injection. Document the current risks.

**Refactor with Safe SQL Practices:** Modify the `get_user` function to use parameterized queries (prepared statements) to prevent SQL injection. This involves changing how the `username` is passed to the SQL query.

**Add Exception Handling:** Implement proper exception handling (try-except-finally blocks) for database operations within the `get_user` function to gracefully handle potential errors and ensure database connections are closed.

**Explain Security Improvements:** Detail each security improvement made during refactoring, explaining how parameterized queries mitigate SQL injection, and how exception handling enhances robustness and resource management.

**Final Task:** Summarize the identified vulnerabilities, the refactored secure solution, and the key reasons for its improved security.

Generating...

check\_circle

Preparing

check\_circle

Identify Security Vulnerabilities

check\_circle

	<p>Refactor with Safe SQL Practices</p> <p>check_circle</p> <p>Add Exception Handling</p> <p>check_circle</p> <p>Explain Security Improvements</p> <p>check_circle</p> <p>Final Task</p> <p>thumb_up</p> <p>thumb_down</p> <p>Show me the refactored `get_user` function</p> <p>Explain how parameterized queries prevent SQL injection</p> <p>How can I apply this SQL security pattern to other database functions</p> <div></div>	
--	--	--