

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	<table border="1"><tr><td>Mr. S Naresh Kumar</td></tr><tr><td>Ms. B. Swathi</td></tr><tr><td>Dr. Sasanko Shekhar Gantayat</td></tr><tr><td>Mr. Md Sallauddin</td></tr><tr><td>Dr. Mathivanan</td></tr><tr><td>Mr. Y Srikanth</td></tr><tr><td>Ms. N Shilpa</td></tr><tr><td>Dr. Rishabh Mittal (Coordinator)</td></tr><tr><td>Dr. R. Prashant Kumar</td></tr><tr><td>Mr. Ankushavali MD</td></tr><tr><td>Mr. B Viswanath</td></tr><tr><td>Ms. Sujitha Reddy</td></tr><tr><td>Ms. A. Anitha</td></tr><tr><td>Ms. M.Madhuri</td></tr><tr><td>Ms. Katherashala Swetha</td></tr><tr><td>Ms. Velpula sumalatha</td></tr><tr><td>Mr. Bingi Raju</td></tr><tr><td>Mr. G. Kranthi</td></tr></table>			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
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	<i>Expect edT ime to co mpl ete</i>
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>	

- 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

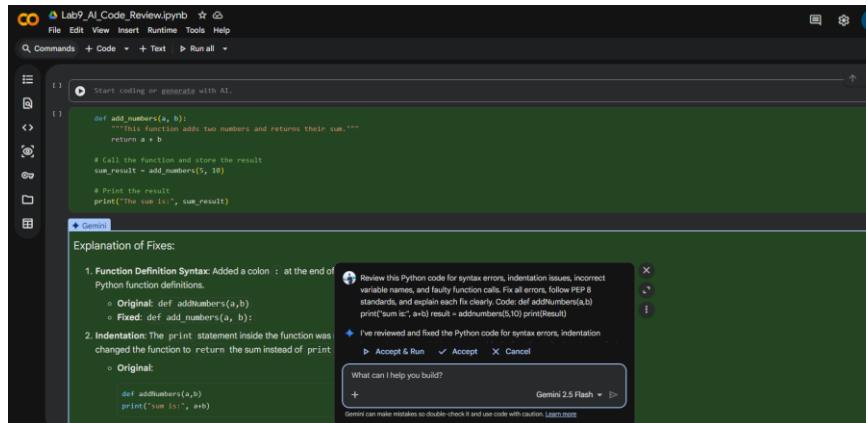
Sample :

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)
```



The screenshot shows a code editor window titled "Lab9_AI_Code_Review.ipynb". The code in the editor is:

```
def add_numbers(a, b):
    """This function adds two numbers and returns their sum."""
    return a + b

# Call the function and store the result
sum_result = add_numbers(5, 10)

# Print the result
print("The sum is:", sum_result)
```

A modal dialog box titled "Explanation of Fixes" is open, listing two types of fixes:

- Function Definition Syntax:** Added a colon : at the end of Python function definitions.
 - Original: `def addNumbers(a,b)`
 - Fixed: `def add_numbers(a, b):`
- Indentation:** The `print` statement inside the function was changed to `return the sum instead of print`.
 - Original:


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

The modal also contains a message from Gemini: "I've reviewed and fixed the Python code for syntax errors, indentation issues, 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)". At the bottom of the modal are buttons for "Accept & Run" and "Accept".

The screenshot shows a code editor interface with an AI code review overlay. The file is named `Lab9_AI_Code_Review.py`. The code contains a function `add_numbers(a, b)` which originally printed the sum instead of returning it. The AI provides explanations for each fix, follows PEP 8 naming conventions, and the code executes successfully.

```

Lab9_AI_Code_Review.py
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
The sum is: 15
Explanation of Fixes:
1. Function Definition Syntax: Added a colon : at the end of the def add_numbers(a, b) line. This is a syntax requirement for Python function definitions.
   - Original: def addnumbers(a,b)
   - Fixed: def add_numbers(a, b):
2. Indentation: The print statement inside the function was not indented. Python uses indentation to define code blocks. I've also changed the function to return the sum instead of printing it, which is generally better practice for reusable functions.
   - Original:
     def addnumbers(a,b)
     print("sum is:", a+b)
   - Fixed:
     def add_numbers(a, b):
         return a + b
3. PEP 8 Naming Convention (Function Name): Changed addnumbers to add_numbers. PEP 8, the style guide for Python, recommends using lowercase words separated by underscores for function and variable names.
   - Original: addnumbers
   - Fixed: add_numbers
4. Faulty Function Call & Variable Assignment: The original addnumbers function only printed the sum but didn't return.

```

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. What can help you build? Gemini 2.5 Flash

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
```

The screenshots illustrate the AI-powered code review process:

- AI Generated Code:** Shows a comparison between the original code (using `result`) and the fixed code (using `sum_result`). It also shows AI suggestions for generating code.
- Task Analysis:** A task titled "Analyze Current Function Performance" is shown. Subtasks include analyzing the time complexity and potential performance bottlenecks of the `find_duplicates` function. Instructions ask for the function's code to proceed with the analysis.
- Performance Optimization:** A summary of key findings from the analysis, including initial bottleneck identification and the introduction of a set to track elements. It highlights the quadratic complexity of the original implementation and the linear complexity of the optimized version using a set. A tooltip provides further details on the performance improvement.

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

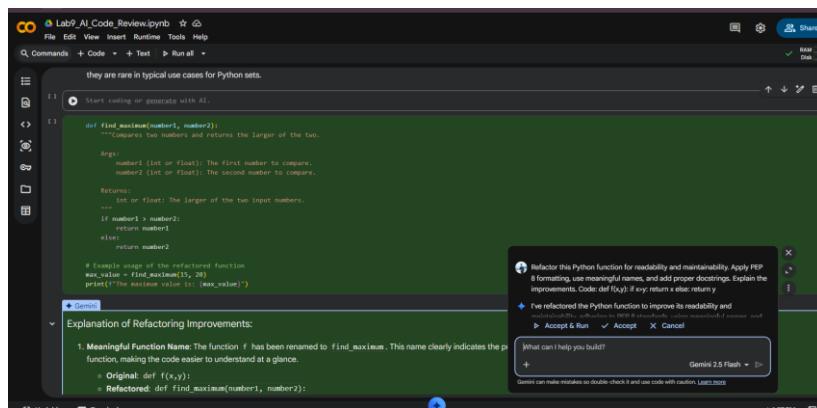
Expected Outcome

- 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
```



The screenshot shows the AI Code Review interface with a Python file named 'Lab9_AI_Code_Review.ipynb'. A tooltip is open over a code block, providing an explanation of refactoring improvements. The tooltip content includes:

- Meaningful Function Name:** The function `f` has been renamed to `find_maximum`. This name clearly indicates the purpose of the function, making the code easier to understand at a glance.
- Original:** `def f(x,y):`
Refactored: `def find_maximum(number1, number2):`
- Meaningful Variable Names:** The parameters `x` and `y` have been renamed to `number1` and `number2`. These names provide better context about the data they represent.
- Original:** `f(x,y)`
Refactored: `find_maximum(number1, number2)`
- Docstring:** A comprehensive docstring has been added to the function. This docstring explains:
 - The function's overall purpose.
 - The Args (arguments/parameters) it expects, including their types and descriptions.
 - The Returns value, including its type and description. Docstrings are crucial for documenting code, especially if that might be reused or maintained by others, as they can be accessed via `help(f) or f.__doc__`.
- PEP 8 Formatting (Indentation and Spacing):** The code has been formatted to comply with PEP 8 standards, specifically indentation (using 4 spaces) and consistent spacing around operators and keywords. This significantly improves visual consistency.
- Original:** Inconsistent indentation (e.g., `if x,y: return x`)
Refactored: Consistent 4-space indentation for code blocks.

The tooltip also notes: "These changes make the function self-documenting to a large extent, reducing the need for extensive comments and ensuring code is understandable and maintainable over time."

'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

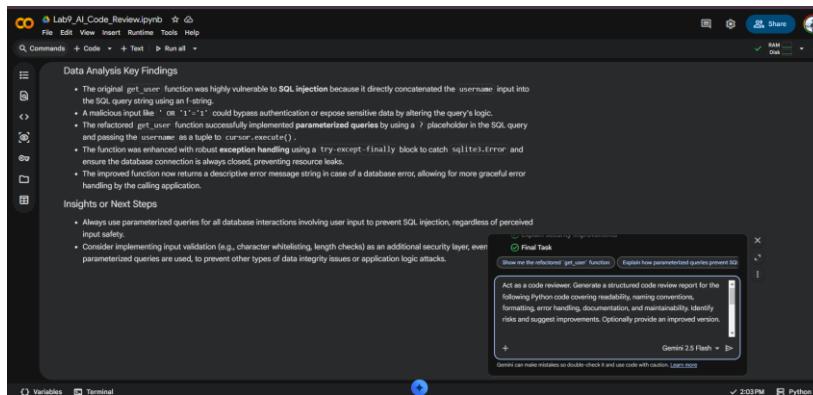
- Production-ready code structure

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
```

The screenshot shows the Code Review interface for a Python file named `Lab9_AI_Code_Review.py`. The main pane displays a task titled "Identify Security Vulnerabilities". A subtask, "Provide `get_user` Function Code", is selected. A modal window titled "Final Task" is open, containing instructions to act as a code reviewer, generate a structured code review report, and provide an improved version of the function. The interface includes standard navigation and status bars at the top and bottom.

The screenshot shows the Code Review interface for the same Python file. The main pane now displays the "Data Analysis Key Findings" section, which lists several security vulnerabilities found in the original `get_user` function. Below this, the "Insights or Next Steps" section provides recommendations for preventing SQL injection. A modal window titled "Final Task" is also present, similar to the one in the first screenshot.

Insights 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

structures (e.g., sets) to improve its time complexity. The optimized code will be provided.

Explain Performance Improvement: 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.

Final Task: Summarize the analysis, the optimized solution, and the key reasons for its improved performance.

Generating...

check_circle

Preparing

check_circle

Analyze Current Function Performance

check_circle

Analyze Current Function Performance (Retry 1)

check_circle

Optimize Function for Time Complexity

history

Explain Performance Improvement

check_circle

Final Task

thumb_up

thumb_down



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

spark

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.

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

Refactor with Safe SQL Practices

check_circle

Add Exception Handling

check_circle

Explain Security Improvements

check_circle

Final Task

thumb_up

thumb_down

Show me the refactored `get_user` function

Explain how parameterized queries prevent SQL injection

How can I apply this SQL security pattern to other database functions

