

## Assignment 10.4 Ai Assisted Coding

Name: MD AMAN

Htno:2303A510C0

Btno:06

### 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

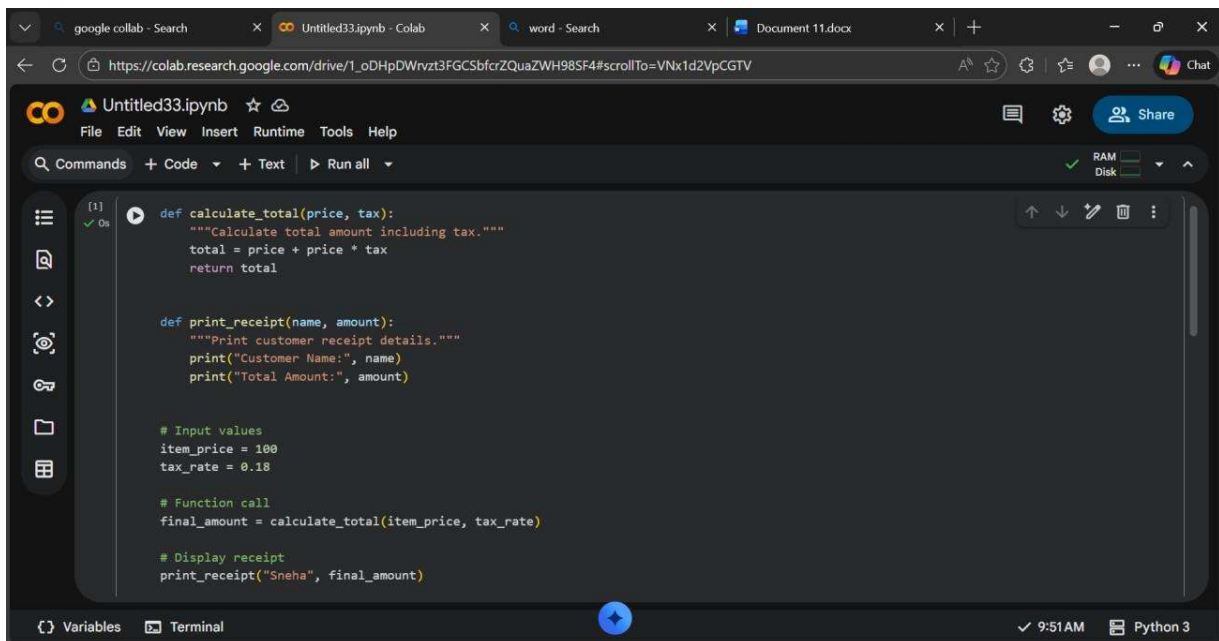
Use an AI tool (GitHub Copilot / Cursor AI) to:

- Identify all syntactic and structural errors
- Correct them systematically
- Generate an explanation of each fix made

#### Expected Outcome

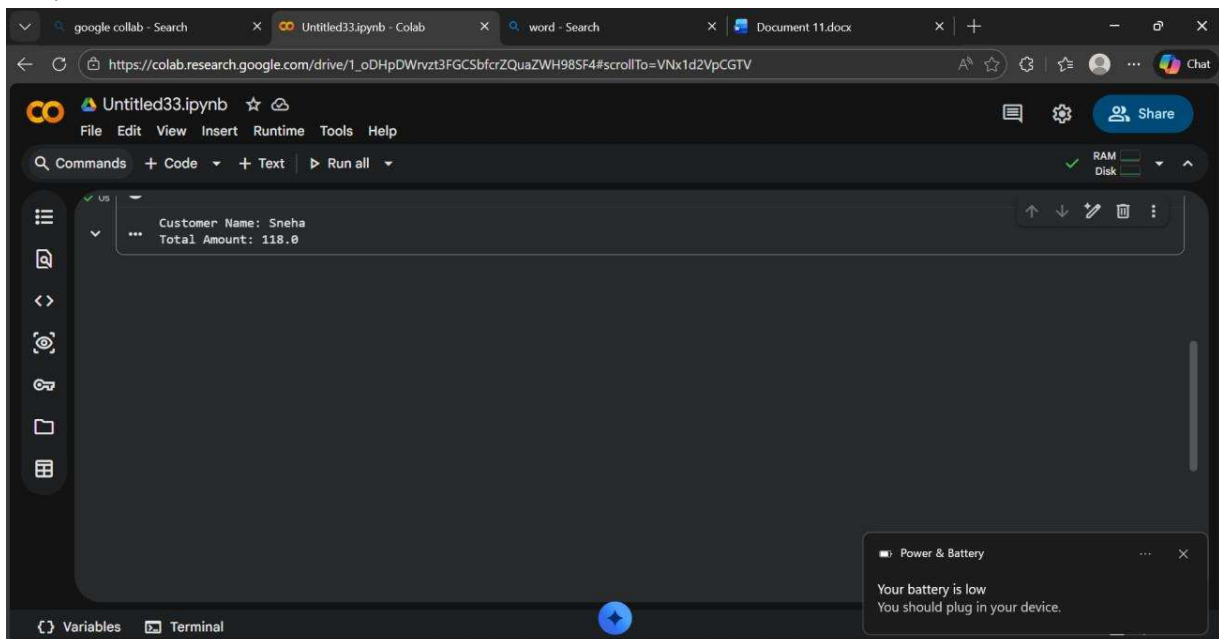
- Fully corrected and executable Python code
- AI-generated explanation describing:
  - o Syntax fixes
  - o Naming corrections
  - o Structural improvements
- Clean, readable version of the script

Code:



```
[1] def calculate_total(price, tax):  
    """Calculate total amount including tax."""  
    total = price + price * tax  
    return total  
  
    def print_receipt(name, amount):  
        """Print customer receipt details."""  
        print("Customer Name:", name)  
        print("Total Amount:", amount)  
  
    # Input values  
    item_price = 100  
    tax_rate = 0.18  
  
    # Function call  
    final_amount = calculate_total(item_price, tax_rate)  
  
    # Display receipt  
    print_receipt("Sneha", final_amount)
```

Output:



```
... Customer Name: Sneha  
... Total Amount: 118.0
```

Explanation:

- >AI fixed syntax mistakes and indentation errors in the script.
- >It corrected wrong function calls and mismatched variable names.
- >Naming was standardized using proper Python conventions.
- >The code structure was cleaned and organized properly.
- >The final program runs correctly without errors.

## 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

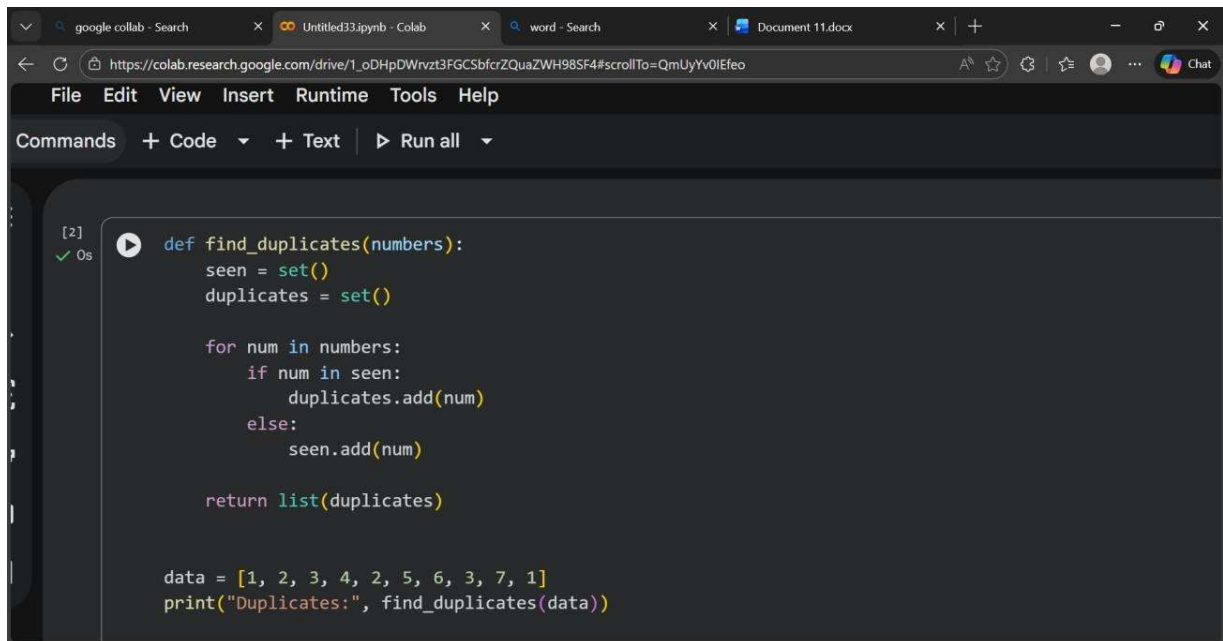
You are provided with a function that identifies duplicate values in a list using inefficient nested loops.

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 Code:



The screenshot shows a Google Colab notebook interface. The top bar includes tabs for 'google collab - Search', 'Untitled33.ipynb - Colab', 'word - Search', and 'Document 11.docx'. The address bar shows the URL 'https://colab.research.google.com/drive/1\_oDHpDWrvzt3FGCSbfcZQuaZWH98SF4#scrollTo=QmUyYv0IEfeo'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu bar, there are tabs for 'Commands', '+ Code', '+ Text', and 'Run all'. The main code cell is labeled '[2]' and shows a Python function 'find\_duplicates' that uses a set to track seen numbers and a list to store duplicates. The function is called with a list of numbers, and the output is printed.

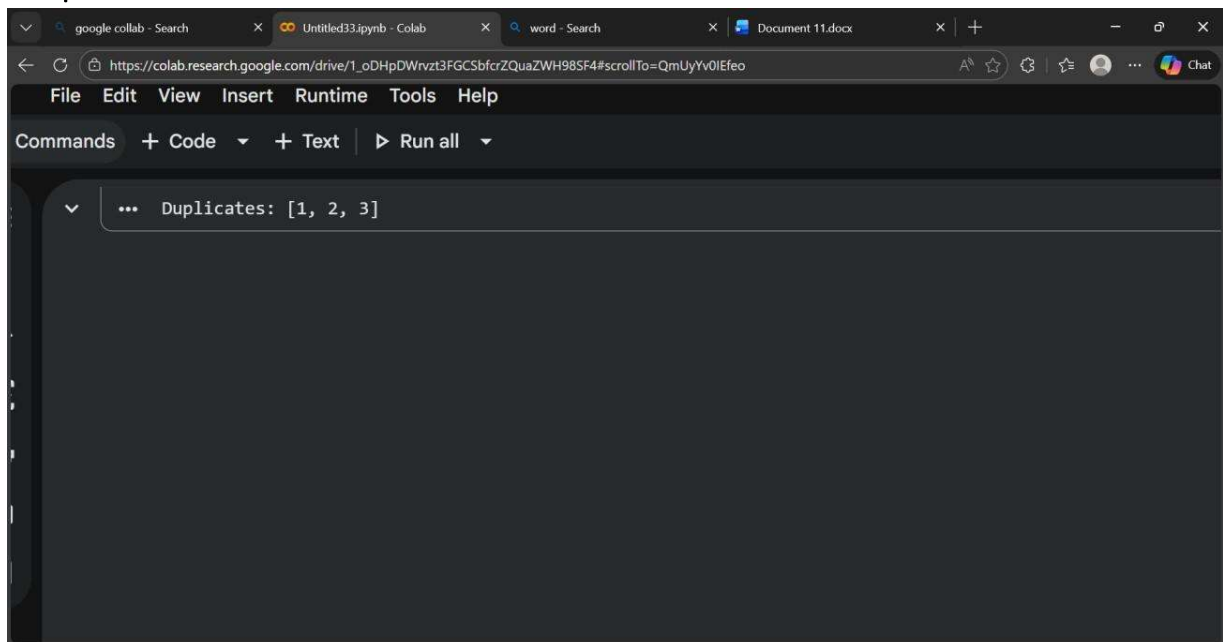
```
[2] def find_duplicates(numbers):
    seen = set()
    duplicates = set()

    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, 1]
print("Duplicates:", find_duplicates(data))
```

Output:



The screenshot shows the same Google Colab notebook interface as the previous one, but now the output of the code cell is visible. The output is 'Duplicates: [1, 2, 3]'. The interface elements are the same as in the previous screenshot.

```
... Duplicates: [1, 2, 3]
```

Explanation:

- >The original code used **nested loops**, comparing each element with every other element.
- >This caused  **$O(n^2)$  time complexity**, making it slow for large lists.
- >The optimized version uses a **set** for quick lookup of seen elements.
- >Set operations work in  **$O(1)$  time**, allowing duplicates to be found in one pass.

-->This reduces overall complexity to  **$O(n)$** , improving performance while keeping correct results.

### 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

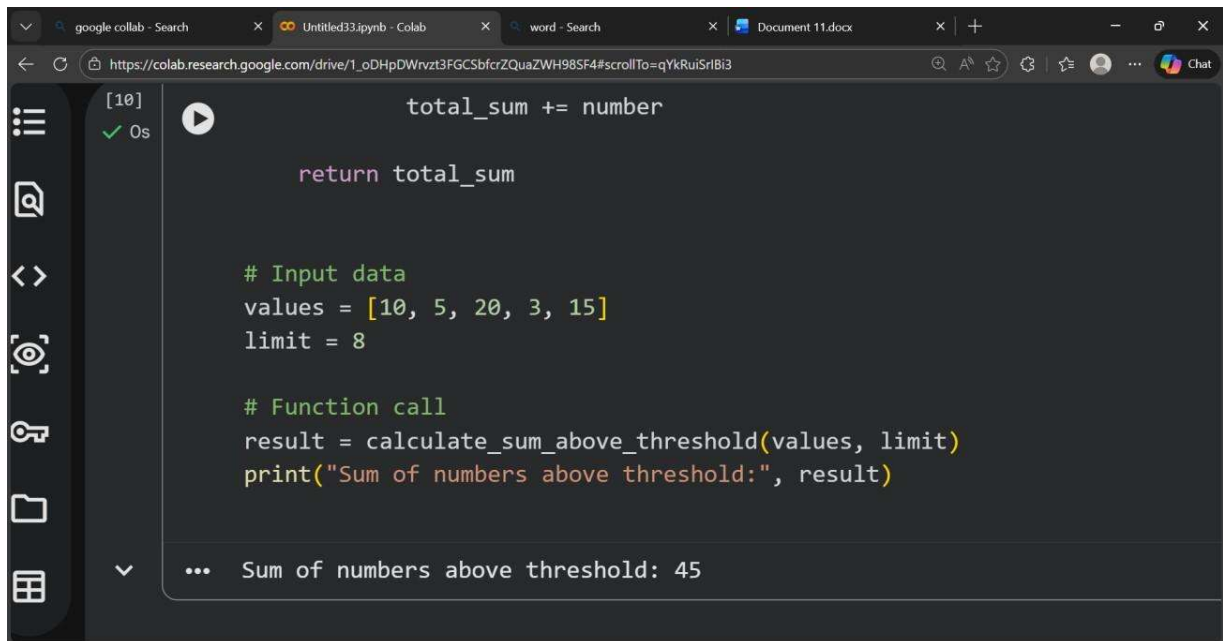
- Docstrings explaining the function purpose
- AI explanation of readability improvements Code:

The image displays two screenshots of a Google Colab notebook interface. The top screenshot shows a Python function named `calculate_sum_above_threshold` with a docstring explaining its purpose and parameters. The bottom screenshot shows the same function being tested with sample data.

```
def calculate_sum_above_threshold(numbers, threshold):  
    """  
    Calculate the sum of numbers greater than a given threshold.  
  
    Parameters:  
    numbers (list): List of numeric values.  
    threshold (int or float): The minimum value to include in the sum.  
  
    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
```

```
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  
  
# Input data  
values = [10, 5, 20, 3, 15]  
limit = 8  
  
# Function call  
result = calculate_sum_above_threshold(values, limit)  
print("Sum of numbers above threshold:", result)
```

Output:



The screenshot shows a Google Colab notebook interface. The top bar includes tabs for 'google collab - Search', 'Untitled33.ipynb - Colab', 'word - Search', and 'Document 11.docx'. The address bar shows a URL from 'https://colab.research.google.com'. The notebook editor has a dark theme. On the left sidebar, there are icons for file explorer, search, and other tools. The main code area contains the following Python code:

```
[10] ✓ 0s
total_sum += number

return total_sum

# Input data
values = [10, 5, 20, 3, 15]
limit = 8

# Function call
result = calculate_sum_above_threshold(values, limit)
print("Sum of numbers above threshold:", result)
```

Below the code, the output is displayed: 'Sum of numbers above threshold: 45'.

## Explanation:

-->The original code was hard to understand due to unclear function and variable names, poor formatting, and no documentation.

--> The refactored version improves readability by using a descriptive function name and meaningful variable names.

-->Proper indentation and spacing were applied following PEP 8 standards. A docstring was added to explain the function's purpose, parameters, and return value.

--> These changes make the code easier to read, maintain, and modify in the future.

## 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

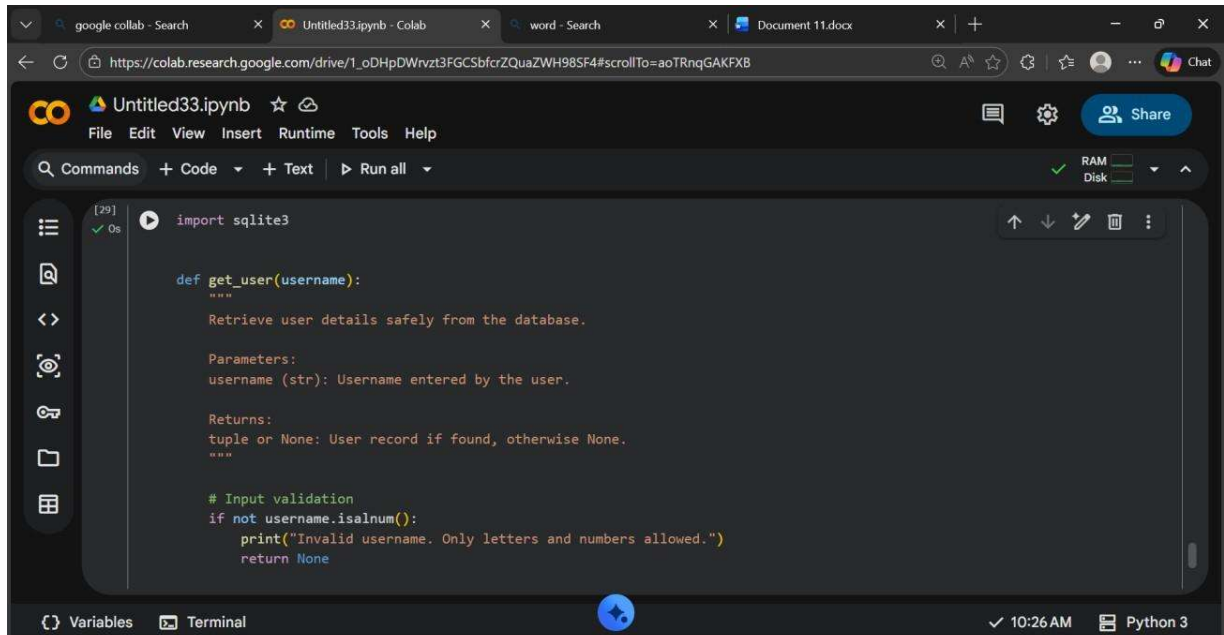
You are given a Python script that:

- 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
- Code:





google collab - Search x Untitled33.ipynb - Colab x word - Search x Document 11.docx

https://colab.research.google.com/drive/1\_oDHpDWrvzt3FGCSbfcZQuaZWH98SF4#scrollTo=aoTRnqGAKFXB

Untitled33.ipynb ☆ Share

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

RAM Disk

```
[29] ✓ Os
import sqlite3

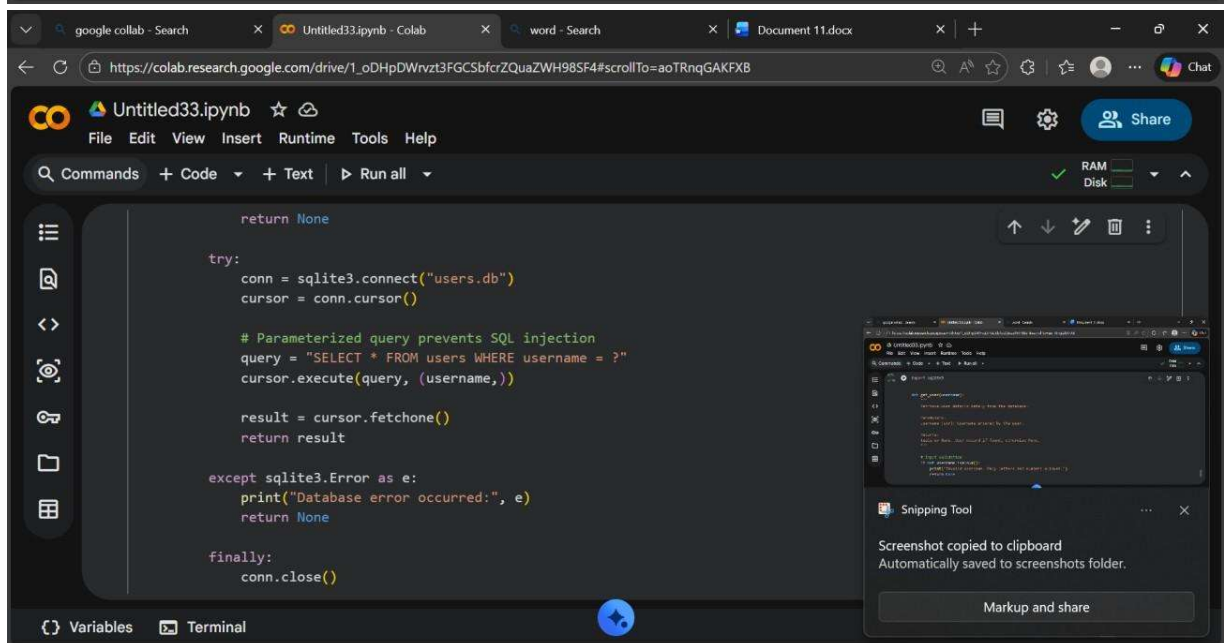
def get_user(username):
    """
    Retrieve user details safely from the database.

    Parameters:
    username (str): Username entered by the user.

    Returns:
    tuple or None: User record if found, otherwise None.
    """

    # Input validation
    if not username.isalnum():
        print("Invalid username. Only letters and numbers allowed.")
        return None
```

Variables Terminal 10:26 AM Python 3



google collab - Search x Untitled33.ipynb - Colab x word - Search x Document 11.docx

https://colab.research.google.com/drive/1\_oDHpDWrvzt3FGCSbfcZQuaZWH98SF4#scrollTo=aoTRnqGAKFXB

Untitled33.ipynb ☆ Share

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

RAM Disk

```
        return None

    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()

        # Parameterized query prevents SQL injection
        query = "SELECT * FROM users WHERE username = ?"
        cursor.execute(query, (username,))

        result = cursor.fetchone()
        return result

    except sqlite3.Error as e:
        print("Database error occurred:", e)
        return None

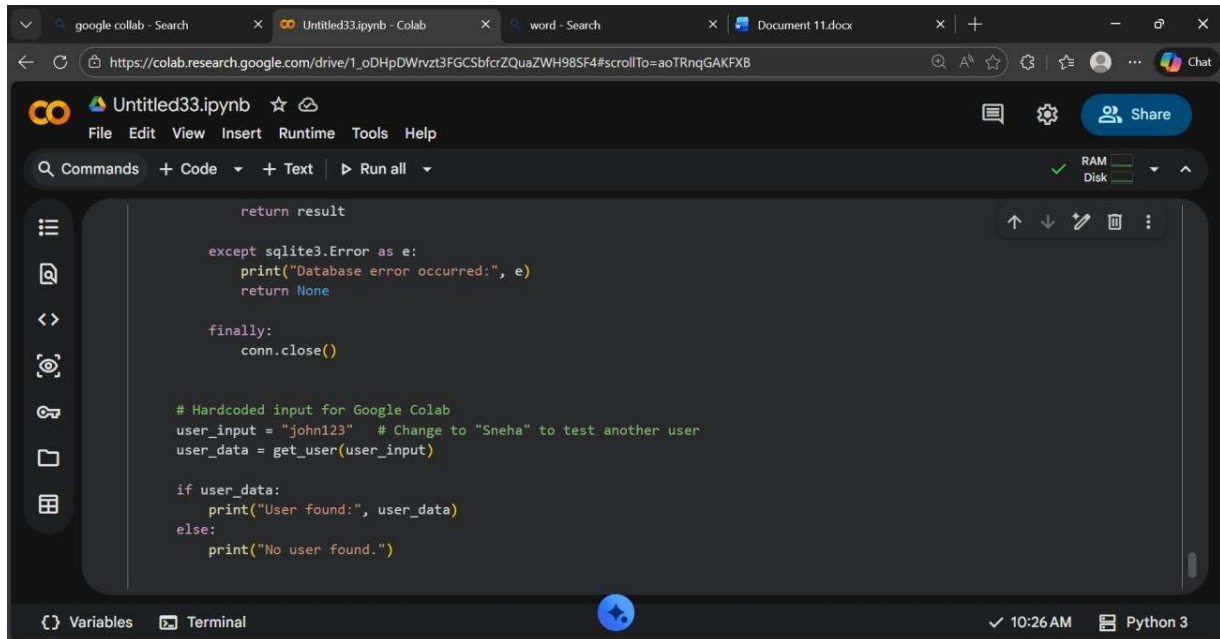
    finally:
        conn.close()
```

Variables Terminal

Snipping Tool

Screenshot copied to clipboard  
Automatically saved to screenshots folder.

Markup and share



The screenshot shows a Google Colab notebook titled 'Untitled33.ipynb'. The code is written in Python and includes a function `get_user` that takes `user_input` as an argument. The function uses a `try-except-finally` block to handle database errors. It also includes a hardcoded input for testing: `user_input = "john123"`. The code prints the user data if found or a message if not found.

```
return result

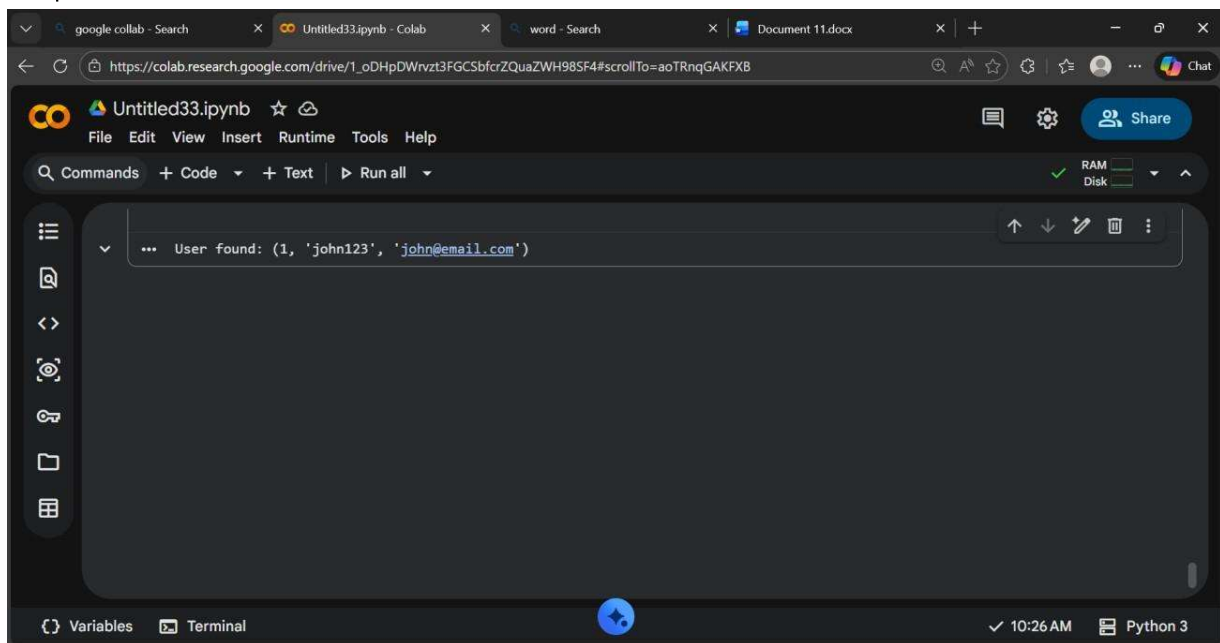
except sqlite3.Error as e:
    print("Database error occurred:", e)
    return None

finally:
    conn.close()

# Hardcoded input for Google Colab
user_input = "john123" # Change to "Sneha" to test another user
user_data = get_user(user_input)

if user_data:
    print("User found:", user_data)
else:
    print("No user found.")
```

Output:



The screenshot shows the same Google Colab notebook, but now the output of the code is visible. The output is a tuple: `(1, 'john123', 'john@email.com')`.

```
... User found: (1, 'john123', 'john@email.com')
```

Explanation:

-->The original code was insecure because it built SQL queries using string concatenation, which could lead to SQL injection attacks.

-->The refactored version uses parameterized queries (?) to safely pass user input to the database.

-->Input validation was added to ensure only alphanumeric usernames are accepted, reducing the risk of malicious input.

-->Try-except blocks were introduced to handle database errors without crashing the program.

--> A finally block ensures the database connection is always closed, improving reliability and making the code safe for production use.

## 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

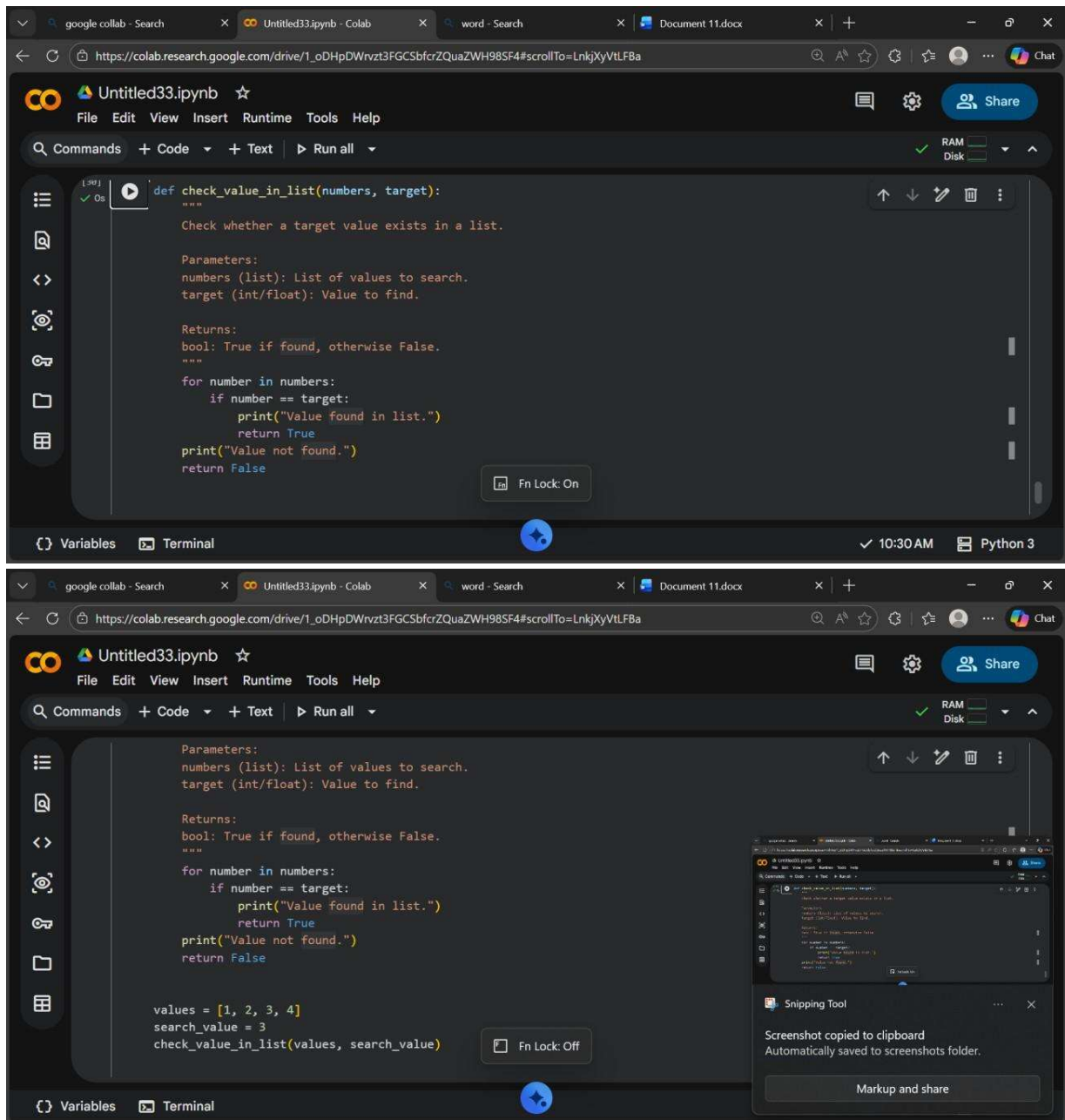
- AI-generated review report including:

o Identified quality issues o Risk areas o

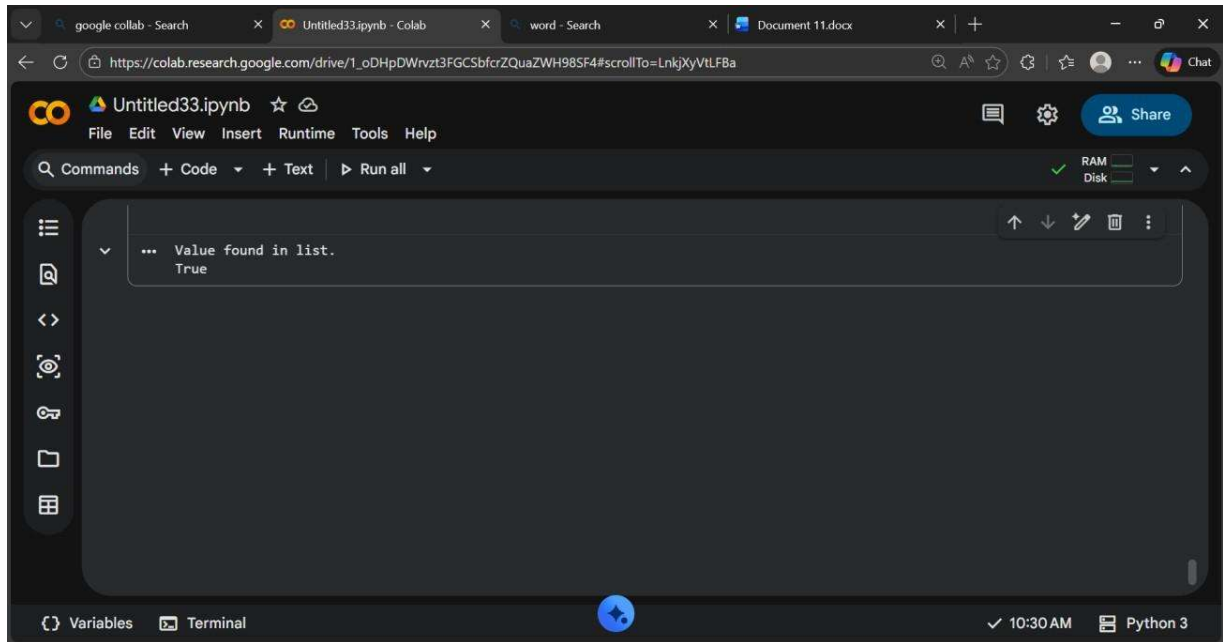
Code smell detection o Improvement

suggestions • Optional improved version  
of the code

Code:



Output:



## Explanation:

-->In this task, AI was used as a code reviewer to analyze code quality instead of just fixing errors.

-->The AI identified issues related to poor readability, unclear naming, bad formatting, missing documentation, and lack of error handling.

-->It also detected code smells such as unused variables and unnecessary statements. Based on this analysis, improvement suggestions were provided to make the code more maintainable and professional.

-->This demonstrates how AI helps teams perform faster and more consistent preliminary code reviews before human evaluation.