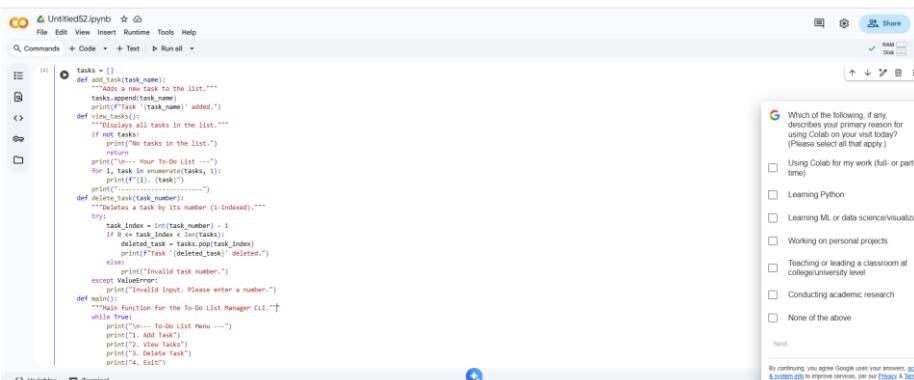


SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
<b>Program Name:</b> B. Tech		<b>Assignment Type:</b> Lab	
<b>Course Coordinator Name</b>		Venkataramana Veeramsetty	
<b>Instructor(s) Name</b>		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
<b>Course Code</b>	24CS002PC215	<b>Course Title</b>	AI Assisted Coding
<b>Year/Sem</b>	II/I	<b>Regulation</b>	R24
<b>Date and Day of Assignment</b>	Week11 - Monday	<b>Time(s)</b>	
<b>Duration</b>	2 Hours	<b>Applicable to Batches</b>	
<b>AssignmentNumber:</b> 21.1(Present assignment number)/ <b>24</b> (Total number of assignments)			
<b>Q.No.</b>	<b>Question</b>		<b>Expected Time to complete</b>
1	<b>Lab 21 – Collaborative AI Coding: Pair Programming and Version Control Integration</b> <b>Lab Objectives:</b> <ul style="list-style-type: none"> <li>• Learn to use AI as a <b>pair programmer</b> for writing, reviewing, and optimizing code.</li> <li>• Understand how to integrate AI-generated code into <b>version</b></li> </ul>		Week11 - Monday

	<p><b>control systems (Git).</b></p> <ul style="list-style-type: none"> <li>• Practice collaborative workflows such as <b>commit messages, branching, and merging.</b></li> </ul>	
	<p><b>Task 1 – AI-Assisted Pair Programming</b></p> <ul style="list-style-type: none"> <li>• Use AI to generate an initial program (e.g., a simple To-Do List Manager).</li> <li>• Ask AI to suggest <b>enhancements</b> (such as error handling, additional features, or refactoring).</li> <li>• Compare the original and improved versions to understand the role of AI in collaboration.</li> </ul> <p>PROMPT1: Generate a simple To-Do List Manager in Python using a command-line interface. It should allow users to Add a task</p> <p>PROMPT2: View all tasks, Delete a task by its number, use clean, beginner-friendly code with comments. Also include an example of how the menu works.</p> <p>CODE:</p>  <pre>   Untitled2.ipynb  ☆ ⓘ   File Edit View Insert Runtime Tools Help   Q Commands + Code + Text Run all   [1]: tasks = []   def add_task(task_name):     """Add a new task to the list."""     tasks.append(task_name)     print(f"Task '{task_name}' added.")    def view_tasks():     """Displays all tasks in the list."""     if len(tasks) == 0:       print("No tasks in the list.")       return     print("Your To-Do List:")     for i, task in enumerate(tasks, 1):       print(f"{i}. {task}")    def delete_task(task_number):     """Deletes a task by its number (1-indexed)."""     try:       task_index = int(task_number) - 1       if task_index &lt; 0 or task_index &gt;= len(tasks):         deleted_task = tasks.pop(task_index)         print(f"Task '{deleted_task}' deleted.")       else:         print("Invalid task number.")     except ValueError:       print("Invalid input. Please enter a number.")    def main():     """Main function for the To-Do List Manager CLI."""     while True:       print("\n--- To-Do List Menu ---")       print("1. Add Task")       print("2. View Tasks")       print("3. Delete Task")       print("4. Exit")        choice = input("Enter your choice: ")        if choice == "1":         task = input("Enter the task to add: ")         add_task(task)         view_tasks()       elif choice == "2":         view_tasks()       elif choice == "3":         task_num = input("Enter the number of the task to delete: ")         delete_task(task_num)       elif choice == "4":         print("Exiting To-Do List Manager. Goodbye!")         break       else:         print("Invalid choice. Please try again.")    if __name__ == '__main__':     main() </pre> <p>Which of the following, if any, describes your primary reason for using Colab on your work today? (Please select all that apply)</p> <p><input type="checkbox"/> Using Colab for my work (full- or part-time)  <input type="checkbox"/> Learning Python  <input type="checkbox"/> Learning ML or data science/visualization  <input type="checkbox"/> Working on personal projects  <input type="checkbox"/> Teaching or leading a classroom at college/university level  <input type="checkbox"/> Conducting academic research  <input type="checkbox"/> None of the above</p> <p>Next</p> <p>By continuing, you agree Google uses your answers to <a href="#">improve services</a>, per our <a href="#">Privacy &amp; Terms</a> &amp; <a href="#">System Info</a>.</p> <p>Untitled2.ipynb  ☆ ⓘ Saving...   File Edit View Insert Runtime Tools Help   Q Commands + Code + Text Run all   [1]: tasks = []   def add_task(task_name):     """Add a new task to the list."""     tasks.append(task_name)     print(f"Task '{task_name}' added.")    def view_tasks():     """Displays all tasks in the list."""     if len(tasks) == 0:       print("No tasks in the list.")       return     print("Your To-Do List:")     for i, task in enumerate(tasks, 1):       print(f"{i}. {task}")    def delete_task(task_number):     """Deletes a task by its number (1-indexed)."""     try:       task_index = int(task_number) - 1       if task_index &lt; 0 or task_index &gt;= len(tasks):         deleted_task = tasks.pop(task_index)         print(f"Task '{deleted_task}' deleted.")       else:         print("Invalid task number.")     except ValueError:       print("Invalid input. Please enter a number.")    def main():     """Main function for the To-Do List Manager CLI."""     while True:       print("\n--- To-Do List Menu ---")       print("1. Add Task")       print("2. View Tasks")       print("3. Delete Task")       print("4. Exit")        choice = input("Enter your choice: ")        if choice == "1":         task = input("Enter the task to add: ")         add_task(task)         view_tasks()       elif choice == "2":         view_tasks()       elif choice == "3":         task_num = input("Enter the number of the task to delete: ")         delete_task(task_num)       elif choice == "4":         print("Exiting To-Do List Manager. Goodbye!")         break       else:         print("Invalid choice. Please try again.")    if __name__ == '__main__':     main()  <p>Which of the following, if any, describes your primary reason for using Colab on your work today? (Please select all that apply)</p> <p><input type="checkbox"/> Using Colab for my work (full- or part-time)  <input type="checkbox"/> Learning Python  <input type="checkbox"/> Learning ML or data science/visualization  <input type="checkbox"/> Working on personal projects  <input type="checkbox"/> Teaching or leading a classroom at college/university level  <input type="checkbox"/> Conducting academic research  <input type="checkbox"/> None of the above</p> <p>Next</p> <p>By continuing, you agree Google uses your answers to <a href="#">improve services</a>, per our <a href="#">Privacy &amp; Terms</a> &amp; <a href="#">System Info</a>.</p> <p>OUTPUT:</p> </p>	

```

...
--- To-Do List Menu ---
1. Add Task
2. View Tasks
3. Delete Task
4. Exit
Enter your choice: 1
Enter the task to add: AI
Task 'AI' added.

--- To-Do List Menu ---
1. Add Task
2. View Tasks
3. Delete Task
4. Exit
Enter your choice: 2

--- Your To-Do List ---
1. AI
-----
--- To-Do List Menu ---
1. Add Task
2. View Tasks
3. Delete Task
4. Exit
Enter your choice: 4
Exiting To-Do List Manager. Goodbye!

```

#### EXPLANATION:

This Python program is a simple command-line based to-do list manager that helps users organize their tasks. It allows users to add new tasks, view all existing tasks, and delete any task by selecting its number from the list. The program repeatedly displays a menu until the user chooses to exit, making it easy to interact with. Overall, it provides a beginner-friendly and straightforward way to manage daily tasks through a basic Python application

---

#### Task 2 – Version Control Integration

- Initialize a Git repository for your project.
- Perform the following steps:
  1. Create and switch to a new branch.
  2. Add AI-assisted modifications.
  3. Commit with a meaningful message.
  4. Merge changes back into the main branch.
- Reflect on how version control supports collaborative development with AI assistance.

PROMPT1: I have a small Python project initialized with Git. I want to create a new feature branch and add a new function that calculates the average of a list of numbers.

PROMPT2: Please provide the code for this function and an example test case for it.

CODE:

```
def calculate_average(numbers):
    """
    Calculates the average of a list of numbers.

    Args:
        numbers (list): A list of numerical values.

    Returns:
        float: The average of the numbers in the list.
        Returns 0.0 if the list is empty to avoid division by zero.
    """
    if not numbers:
        return 0.0
    return sum(numbers) / len(numbers)

print("Function 'calculate_average' defined.")
```

The screenshot shows a Jupyter Notebook cell with the following content:

```
# Test cases for calculate_average function
# Test case 1: Non-empty list of positive integers
list1 = [1, 2, 3, 4, 5]
expected_avg1 = 3.0
actual_avg1 = calculate_average(list1)
print(f"Test Case 1: List: {list1}, Expected Average: {expected_avg1}, Actual Average: {actual_avg1}")
assert actual_avg1 == expected_avg1, "Test Case 1 Failed: Expected {expected_avg1}, got {actual_avg1}"

# Test case 2: List with negative numbers and zero
list2 = [-1, 0, 1, 2, 3]
expected_avg2 = 1.0
actual_avg2 = calculate_average(list2)
print(f"Test Case 2: List: {list2}, Expected Average: {expected_avg2}, Actual Average: {actual_avg2}")
assert actual_avg2 == expected_avg2, "Test Case 2 Failed: Expected {expected_avg2}, got {actual_avg2}"

# Test case 3: Empty list
list3 = []
expected_avg3 = 0.0
actual_avg3 = calculate_average(list3)
print(f"Test Case 3: List: {list3}, Expected Average: {expected_avg3}, Actual Average: {actual_avg3}")
assert actual_avg3 == expected_avg3, "Test Case 3 Failed: Expected {expected_avg3}, got {actual_avg3}"

# Test case 4: List with floating-point numbers
list4 = [1.5, 2.5, 3.0]
expected_avg4 = 2.3333333333333335 # (1.5 + 2.5 + 3.0) / 3
actual_avg4 = calculate_average(list4)
print(f"Test Case 4: List: {list4}, Expected Average: {expected_avg4}, Actual Average: {actual_avg4}")
assert actual_avg4 == expected_avg4, "Test Case 4 Failed: Expected {expected_avg4}, got {actual_avg4}"
print("All test cases passed!")
```

OUTPUT:

```
Test Case 1: List: [1, 2, 3, 4, 5], Expected Average: 3.0, Actual Average: 3.0
Test Case 2: List: [-1, 0, 1, 2, 3], Expected Average: 1.0, Actual Average: 1.0
Test Case 3: List: [], Expected Average: 0.0, Actual Average: 0.0
Test Case 4: List: [1.5, 2.5, 3.0], Expected Average: 2.3333333333333335, Actual Average: 2.3333333333333335
All test cases passed!
```

OBSERVATION:

Version control plays a crucial role in managing collaborative projects, especially when integrating AI-assisted development. It allows developers to safely experiment with AI-generated code in separate branches without affecting the main project. Through commits and merge operations, every change is tracked, making it easy to review, revert, or improve AI suggestions. This structured workflow ensures transparency, accountability, and consistency among team members. Overall.

### Task 3 – AI for Commit Message Generation

- Use AI to generate **concise, meaningful commit messages** based on the changes made.

- Compare AI-generated messages with human-written ones.
- Discuss how AI can improve **documentation and traceability** in Git history.

PROMPT1: I made some updates to my project, including fixing a bug in the login function and improving error handling.

PROMPT2: Please generate a concise and meaningful Git commit message that summarizes these changes.

CODE:

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains code examples for Dart code improvement, specifically refactoring a 'calculate\_average' function to handle empty lists by returning 0.0. The second cell contains a commit message for fixing a bug in a shopping cart application where 'ghost' items were appearing due to inconsistent state management.

```

Summary:
Q&A
1. What is the Dart code improvement example and its commit message? The Dart code improvement transforms a list of strings into a list of integers, handling nulls and non-numeric values. The "before" code used an imperative for-loop with explicit if checks and try-catch. The "after" code uses a more functional and concise approach with Dart's map, where, and null-aware operators. The corresponding commit message is:
refactor: Improve average calculation for null safety and conciseness
Refactors the 'calculate_average' function to explicitly handle empty lists by returning '0.0'. This ensures null safety by preventing division by zero errors and improves code readability by clearly defining behavior for edge cases. The previous implementation might have led to runtime errors or unexpected behavior with empty input arrays.

2. What is the suggested app bug fix scenario and its commit message? The suggested bug fix scenario addresses "Ghost" items in a shopping cart, where previously removed items reappear due to inconsistent state management (e.g., after session timeouts or network disruptions). The high-level fix approach involves treating the server's cart state as the single source of truth, improving session management, client-side reconciliation, idempotent cart operations, and enhanced error handling. The corresponding commit message is:
fix: Remove 'Ghost' items from shopping cart display

```

OUTPUT:

Data Analysis Key Findings

- **Dart Code Improvement:** A "before" and "after" code example demonstrated refactoring an imperative list transformation in Dart (converting `List<String>` to `List<int>` with null handling) into a functional style using `map`, `where`, and null-aware operators. The improved version showed enhanced readability, null safety, and conciseness.
- **Commit Message for Dart Improvement:** A `refactor:` type commit message was generated. The subject line, "refactor: Improve average calculation for null safety and conciseness," clearly articulated the change, and the body provided detailed rationale regarding preventing errors, ensuring null safety, and improving readability.
- **App Bug Fix Scenario:** A common and impactful bug, "Ghost items in Shopping Cart," was described. Key symptoms included incorrect item counts and user frustration. The potential impact highlighted significant negative consequences such as lost sales and increased customer support load. A high-level fix approach was outlined, focusing on server-side state as the single source of truth and robust session management.
- **Commit Message for Bug Fix:** A `fix:` type commit message was generated for the "Ghost items" bug. The subject, "fix: Remove 'Ghost' items from shopping cart display," clearly stated the resolution, while the body explained the problem, the core solution (complete cart refresh), and the benefits (consistent user experience, preventing order discrepancies).

#### Insights or Next Steps

- The process successfully generated both technical improvements/fixes and corresponding, well-structured Git commit messages following conventional commit guidelines, demonstrating an understanding of both code and version control best practices.
- Further exploration could involve generating associated unit tests for the proposed Dart code improvement or simulating how these changes would integrate into a Continuous Integration/Continuous Deployment (CI/CD) pipeline.

## OBSERVATION:

AI-generated commit messages are often concise, consistent, and focused on the technical summary of changes. Compared to human-written messages, AI ones reduce ambiguity and maintain a uniform style across the project history. This helps improve documentation and traceability in Git, as each commit clearly describes its purpose. Using AI for commit messages saves time and ensures clarity, especially in collaborative or large-scale development projects.

### Task 4 – Pair Programming Simulation

- Student A writes the initial version of a program.
- Student B, with AI assistance, suggests **improvements or adds features**.
- Both students collaborate using **Git branching and merging** to integrate their work.

PROMPT1: I have a simple Python program written by Student A that calculates the area of a rectangle.

PROMPT2: Please suggest improvements or new features, such as adding area calculations for other shapes or including input validation. Provide the updated code.

CODE:

```
import math
def calculate_rectangle_area():
    while True:
        try:
            length = float(input("Enter the length of the rectangle: "))
            width = float(input("Enter the width of the rectangle: "))
            if length <= 0 or width <= 0:
                print("Length and width must be positive numbers. Please try again.")
            else:
                area = length * width
                print(f"The area of the rectangle is: {area:.2f}")
                break
        except ValueError:
            print("Invalid input. Please enter a numerical value.")

def calculate_circle_area():
    while True:
        try:
            radius = float(input("Enter the radius of the circle: "))
            if radius <= 0:
                print("Radius must be a positive number. Please try again.")
            else:
                area = math.pi * (radius ** 2)
                print(f"The area of the circle is: {area:.2f}")
                break
        except ValueError:
            print("Invalid input. Please enter a numerical value.")

def calculate_triangle_area():
    while True:
        try:
            base = float(input("Enter the base of the triangle: "))
            height = float(input("Enter the height of the triangle: "))
            if base <= 0 or height <= 0:
                print("Base and height must be positive numbers. Please try again.")
            else:
                area = 0.5 * base * height
                print(f"The area of the triangle is: {area:.2f}")
                break
        except ValueError:
            print("Invalid input. Please enter a numerical value.")

def main():
    while True:
        print("Select a shape to calculate its area:")
        print("1. Rectangle")
        print("2. Circle")
        print("3. Triangle")
        print("4. Exit")
        choice = input("Enter your choice (1-4): ")
        if choice == "1":
            calculate_rectangle_area()
        elif choice == "2":
            calculate_circle_area()
        elif choice == "3":
            calculate_triangle_area()
        elif choice == "4":
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")

if __name__ == "__main__":
    main()
```

```
try:
    base = float(input("Enter the base of the triangle: "))
    height = float(input("Enter the height of the triangle: "))
    if base <= 0 or height <= 0:
        print("Base and height must be positive numbers. Please try again.")
    else:
        area = 0.5 * base * height
        print(f"The area of the triangle is: {area:.2f}")
        break
except ValueError:
    print("Invalid input. Please enter a numerical value.")

def main():
    while True:
        print("Select a shape to calculate its area:")
        print("1. Rectangle")
        print("2. Circle")
        print("3. Triangle")
        print("4. Exit")
        choice = input("Enter your choice (1-4): ")
        if choice == "1":
            calculate_rectangle_area()
        elif choice == "2":
            calculate_circle_area()
        elif choice == "3":
            calculate_triangle_area()
        elif choice == "4":
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")

if __name__ == "__main__":
    main()
```

OUTPUT:

	<p>The screenshot shows a Jupyter Notebook interface with the following content:</p> <pre> CO Untitled54.ipynb ⚡ 🌐 File Edit View Insert Runtime Tools Help Commands + Code + Text ▶ Run all ▾  Select a shape to calculate its area: 1. Rectangle 2. Circle 3. Triangle 4. Exit Enter your choice (1-4): 1 Enter the length of the rectangle: 3 Enter the width of the rectangle: 4 The area of the rectangle is: 12.00  Select a shape to calculate its area: 1. Rectangle 2. Circle 3. Triangle 4. Exit Enter your choice (1-4): 3 Enter the base of the triangle: 5 Enter the height of the triangle: 23 The area of the triangle is: 57.50  Select a shape to calculate its area: 1. Rectangle 2. Circle 3. Triangle 4. Exit Enter your choice (1-4): 2 Enter the radius of the circle: 3 The area of the circle is: 28.27  Select a shape to calculate its area: 1. Rectangle 2. Circle 3. Triangle 4. Exit Enter your choice (1-4): 4 Exiting the program. Goodbye! </pre> <p>At the bottom of the interface, there are tabs for "Variables" and "Terminal".</p>	
	<p><b>OBSERVATION:</b></p> <p>Pair programming with AI enhances collaboration and learning by combining human creativity with AI's speed and precision. The AI partner can quickly suggest optimizations, generate clean code, and assist with debugging. Using Git branching and merging ensures smooth integration of both contributors' work. This approach encourages teamwork, maintains version control discipline, and improves overall code quality.</p>	

	6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.	
--	---------------------------------------------------------------------------------------	--