

AI ASSISTED CODING

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

SHASHANK

2303A510i3

BATCH – 03

16 – 01 – 2026

ASSIGNMENT – 2.5

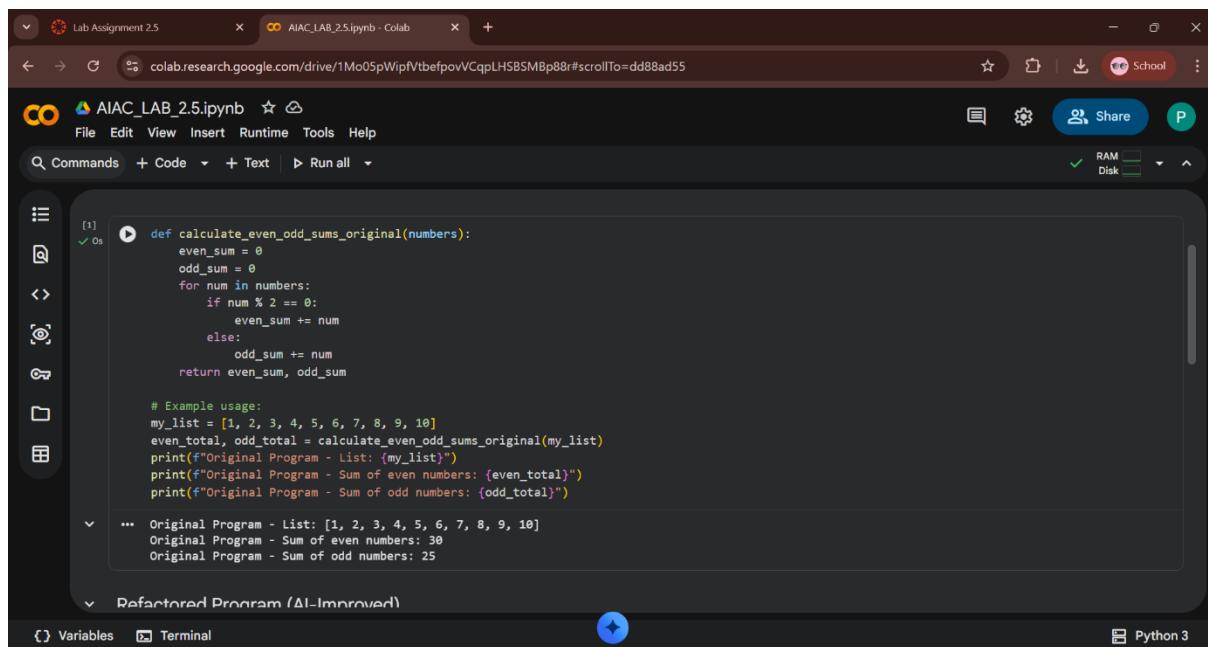
Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI.

Task 1: Refactoring Odd/Even Logic (List Version).

Prompt: Write a Python program that takes a list of integers and calculates the sum of even numbers and the sum of odd numbers separately. After generating the working program, refactor (improve) the code using AI to make it cleaner, more readable, and efficient. Provide both the original code and the refactored version.

Code:

Original Code:



The screenshot shows a Google Colab interface with the following code in cell [1]:

```
[1]: def calculate_even_odd_sums_original(numbers):
    even_sum = 0
    odd_sum = 0
    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return even_sum, odd_sum

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_total, odd_total = calculate_even_odd_sums_original(my_list)
print(f"Original Program - List: {my_list}")
print(f"Original Program - Sum of even numbers: {even_total}")
print(f"Original Program - Sum of odd numbers: {odd_total}")

... Original Program - List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Original Program - Sum of even numbers: 30
Original Program - Sum of odd numbers: 25
```

Below the code, a section titled "Refactored Program (AI-Improved)" is shown, but no code is visible in that section.

AI Improved Code:

The screenshot shows a Jupyter Notebook interface with two tabs: "Lab Assignment 2.5" and "AIAC_LAB_2.5.ipynb". The "AIAC_LAB_2.5.ipynb" tab is active, displaying the following Python code:

```
[2] ✓ Os
def calculate_even_odd_sums_refactored(numbers):
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    return even_sum, odd_sum

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_total_refactored, odd_total_refactored = calculate_even_odd_sums_refactored(my_list)
print(f"\nRefactored Program - List: {my_list}")
print(f"Refactored Program - Sum of even numbers: {even_total_refactored}")
print(f"Refactored Program - Sum of odd numbers: {odd_total_refactored}")

...
Refactored Program - List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Refactored Program - Sum of even numbers: 30
Refactored Program - Sum of odd numbers: 25
```

The code has been refactored to use list comprehensions and return multiple values. The output shows the original list and the calculated sums for even and odd numbers.

Task 2: Area Calculation Explanation.

Prompt: Write a Python function that calculates the area of different shapes (circle, square, and rectangle) based on user input. Then explain the function line by line in simple language so that a junior developer can easily understand it.

Code:

The screenshot shows a Jupyter Notebook interface with two tabs: "Lab Assignment 2.5" and "AIAC_LAB_2.5.ipynb". The "AIAC_LAB_2.5.ipynb" tab is active, displaying the following Python code with explanatory comments:

```
Refactored Program - List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Refactored Program - Sum of even numbers: 30
Refactored Program - Sum of odd numbers: 25

# Import math module
def calculate_area():
    """
    Calculates the area of a circle, square, or rectangle based on user input.
    """
    shape = input("Enter the shape (circle, square, rectangle): ").lower()

    if shape == "circle":
        radius = float(input("Enter the radius of the circle: "))
        if radius <= 0:
            print("Radius cannot be negative.")
        else:
            area = math.pi * radius ** 2
            print(f"The area of the circle is: {area:.2f}")
    elif shape == "square":
        side = float(input("Enter the side length of the square: "))
        if side <= 0:
            print("Side length cannot be negative.")
        else:
            area = side * 2
            print(f"The area of the square is: {area:.2f}")
    elif shape == "rectangle":
        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 cannot be negative.")
        else:
            area = length * width
            print(f"The area of the rectangle is: {area:.2f}")
    else:
        print("Invalid shape. Please choose from 'circle', 'square', or 'rectangle'.")

# Call the function to test it
calculate_area()

-- Enter the shape (circle, square, rectangle): circle
Enter the radius of the circle: 22
The area of the circle is: 502.49
```

The code defines a function `calculate_area()` that prompts the user to enter the shape (circle, square, or rectangle). It then asks for the necessary dimensions (radius for circle, side length for square, and both length and width for rectangle) and calculates the area using the appropriate formula. The output shows a test run for a circle with a radius of 22.

Explanation:

The calculate_area() function helps users find the area of circles, squares, or rectangles. It asks for the shape and necessary dimensions, like radius or side length, then calculates and prints the result. The function includes error handling to manage invalid inputs and ensures dimensions are non-negative, using Python's math module for calculations.

Task 3: Prompt Sensitivity Experiment.

Prompt 1(Simple): Write a Python Code to check whether the given Number is Prime or Not

Code:

The screenshot shows a code editor interface with multiple tabs. The main tab contains the following Python code:

```
def is_prime(n):
    """
    Check if a number is prime.
    """
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True

if __name__ == "__main__":
    try:
        number = int(input("Enter a number to check if it's prime:"))
        if is_prime(number):
            print(f"{number} is a prime number.")
        else:
            print(f"{number} is not a prime number.")
    except ValueError:
        print("Invalid input! Please enter a valid integer.")
    except Exception as e:
        print(f"An error occurred: {e}")

```

Prompt 2(Medium & Detailed): Write an efficient Python function that checks whether a number is prime. Use proper variable names and handle edge cases.

Code:

The screenshot shows the AIAC_1121 interface with a task titled "Python prime number check". The task details ask for an efficient Python function that checks whether a number is prime, using proper variable names and handling edge cases. The code editor contains a Python script named "Lab_2.5.py" with the following content:

```
def is_prime(number):
    if number < 2:
        return False
    if number == 2:
        return True
    if number % 2 == 0:
        return False
    divisor = 3
    while divisor * divisor <= number:
        if number % divisor == 0:
            return False
        divisor += 2
    return True

if __name__ == "__main__":
    try:
        number = int(input("Enter a number to check if it's prime:"))
        if is_prime(number):
            print(f"{number} is a prime number.")
        else:
            print(f"{number} is not a prime number.")

    except ValueError:
        print("Invalid input! Please enter a valid integer.")
    except Exception as e:
        print(f"An error occurred: {e}")

```

The interface includes a sidebar with "Agents" and a bottom status bar indicating "No connection".

Prompt 3(Hard & More Detailed): Write a Python function to check prime numbers using a loop. Avoid built-in functions, add proper comments, and follow clean coding practices.

Code:

The screenshot shows the AIAC_1121 interface with a task titled "Python prime number check". The task details ask for a Python function to check prime numbers using a loop, avoiding built-in functions, adding proper comments, and following clean coding practices. The code editor contains a Python script named "Lab_2.5.py" with the following content:

```
def is_prime(number):
    """
    Check if a number is prime using a loop-based approach.

    A prime number is a natural number greater than 1 that has no divisors other than 1 and itself.

    Args:
        number (int): The number to check for primality

    Returns:
        bool: True if number is prime, False otherwise
    """

    # Edge case: numbers less than 2 cannot be prime
    # (including negative numbers, zero, and one)
    if number < 2:
        return False

    # Edge case: 2 is the smallest and only even prime number
    if number == 2:
        return True

    # Edge case: all even numbers greater than 2 are composite
    # (divisible by 2, so not prime)
    if number % 2 == 0:
        return False

    # Check for divisors using a loop
    # Start from 3 and check only odd numbers (even numbers
    # are divisible by 2, so not prime)
    # We only need to check up to the square root of the number
    # because if number has a factor greater than sqrt(number)
    # then it must have a corresponding factor less than sqrt(number)

```

The interface includes a sidebar with "Agents" and a bottom status bar indicating "No connection".

Task 4: Tool Comparison Reflection.

Prompt: Based on my experiments with Gemini, GitHub Copilot, and Cursor AI, compare these three tools in terms of:

- Ease of use
 - Code quality
 - Clarity of explanation
 - Overall usefulness for students and developers
- Write a short reflection (8–10 lines).

Short Reflection:

Based on my experience, all three AI tools—Gemini, GitHub Copilot, and Cursor AI—are useful for AI-assisted coding, but in different ways. Gemini was very helpful for understanding concepts because it provided clear explanations along with code. GitHub Copilot generated concise and professional-level code, making it suitable for real-world development. However, Copilot gave limited explanations compared to Gemini. Cursor AI was useful for experimenting with different prompts and observing how code changes based on instructions. It also helped in refactoring and improving existing code effectively. In terms of usability, Gemini was the most beginner-friendly, while Copilot and Cursor were better suited for experienced programmers. Overall, Gemini is best for learning, Copilot for coding speed, and Cursor AI for refinement and experimentation.

THANK YOU !!