AI ASSISTED CODING

Nasani Sai Pavan **2303A51117**

BATCH – 03 **16 – 01 – 2026**

ASSIGNMENT – 2.5

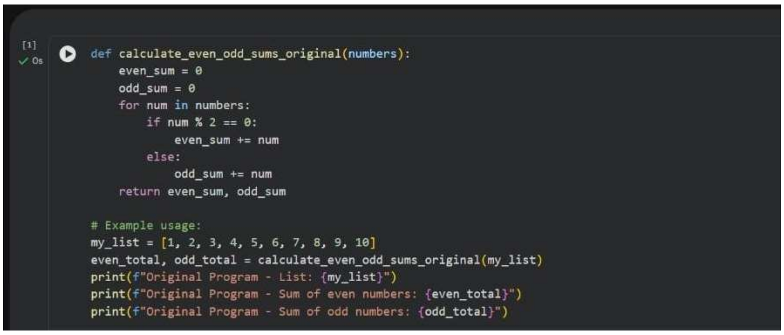
Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI.

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.

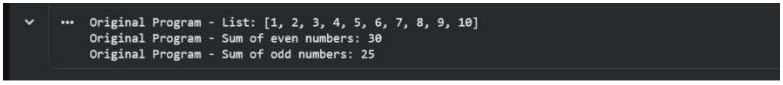
Code:

Original Code:



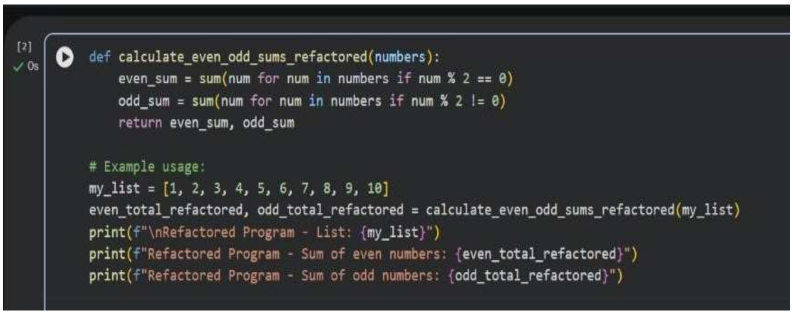
```python
def calculate_even_odd_sums_original(numbers):
    even_sum = 0
    odd_sum = 0
    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return even_sum, odd_sum

# 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}")
```

Output:



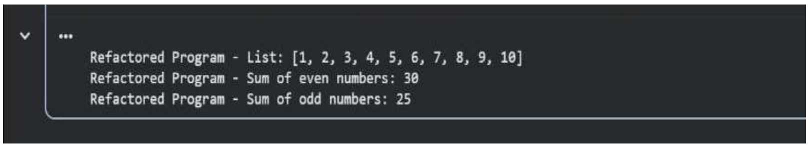
```
Original Program - List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Original Program - Sum of even numbers: 30
Original Program - Sum of odd numbers: 25
```

AI Improved Code:



```python
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}")
```

Output:

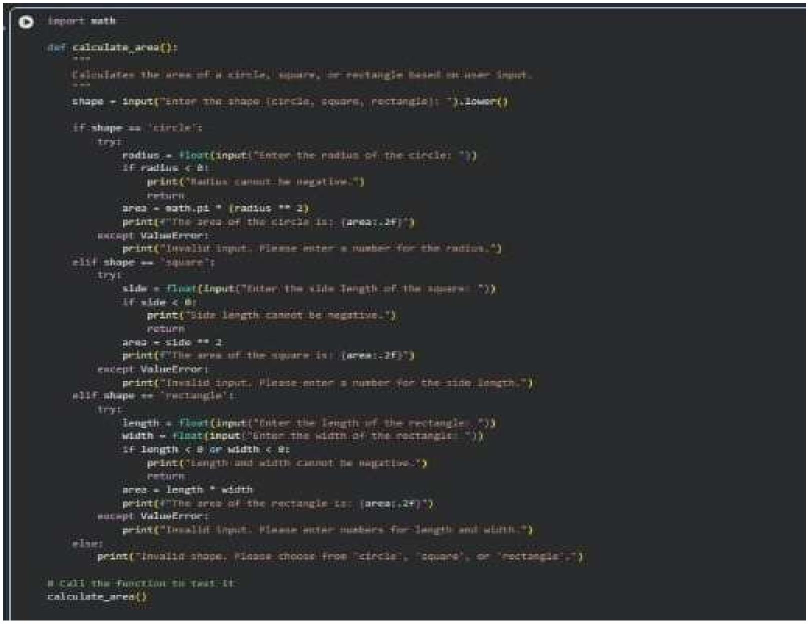


```
Refactored Program - List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Refactored Program - Sum of even numbers: 30
Refactored Program - Sum of odd numbers: 25
```

Task 2: Area Calculation Explanation.

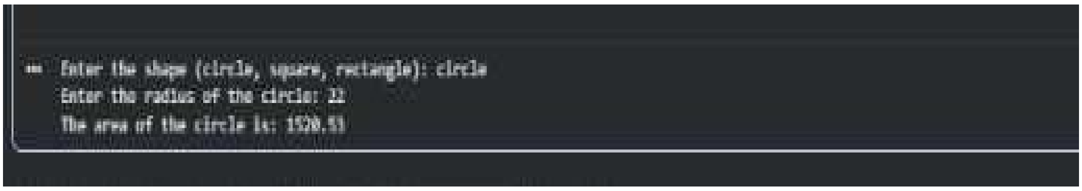
Prompt: Write a Python function that calculates the area of different shapes (circle, square, and rectangle) based on user input. Then explain the function line by line in simple language so that a junior developer can easily understand it.

Code:



```python
import math

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

# Call the function to test it
calculate_area()
```

Output:



```
Enter the shape (circle, square, rectangle): circle
Enter the radius of the circle: 22
The area of the circle is: 1520.53
```

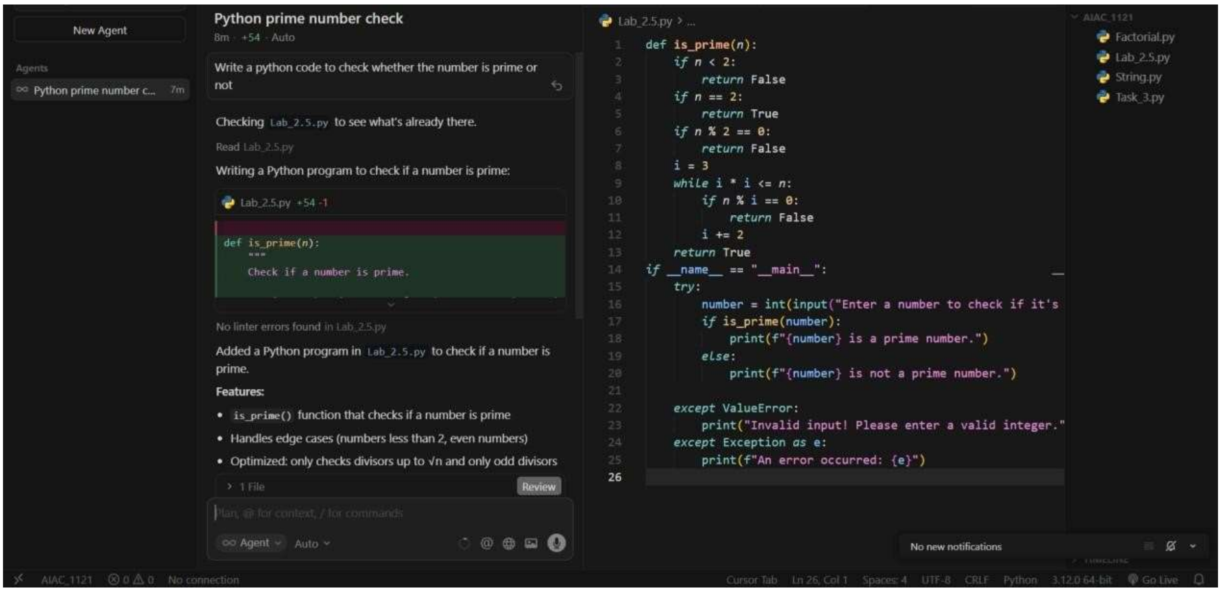
Explanation:

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:



Python prime number check

Write a python code to check whether the number is prime or not

Checking `Lab_2.5.py` to see what's already there.

Read Lab_2.5.py

Writing a Python program to check if a number is prime:

```
def is_prime(n):
    """
    Check if a number is prime.
```

No linter errors found in Lab_2.5.py

Added a Python program in `Lab_2.5.py` to check if a number is prime.

Features:

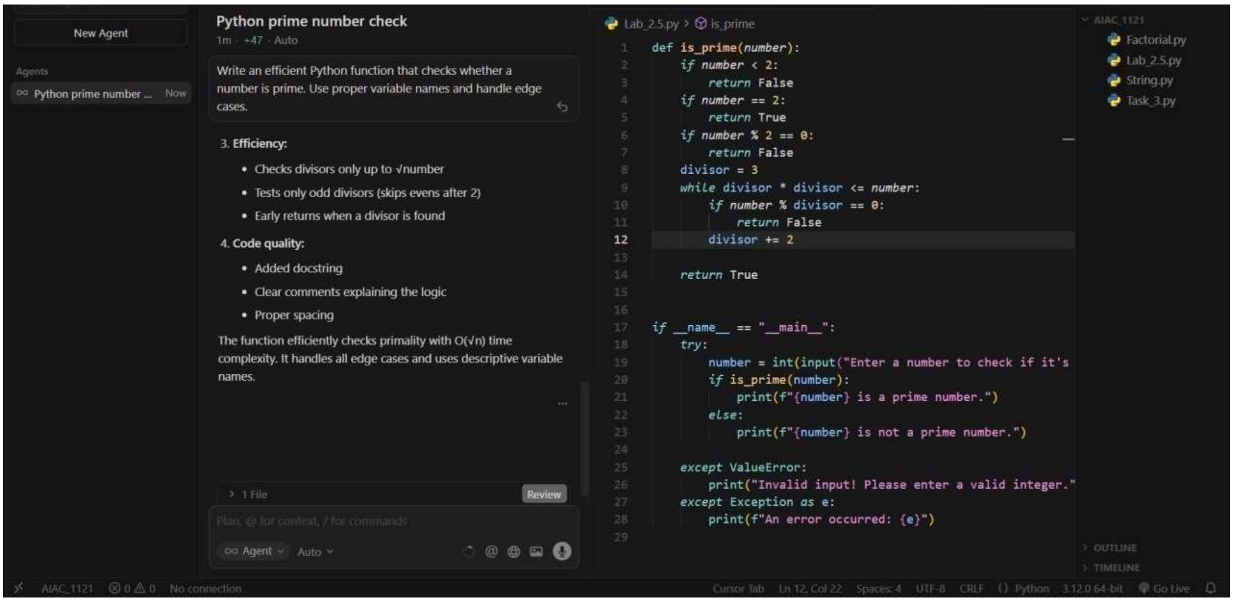
- `is_prime()` function that checks if a number is prime
- Handles edge cases (numbers less than 2, even numbers)
- Optimized: only checks divisors up to √n and only odd divisors

```
def is_prime(n):
    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
        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}")

```

AIAC_1121
Factorial.py
Lab_2.5.py
String.py
Task_3.py
No new notifications
AIAC_1121 ⊗ 0 ⚠ 0 No connection
Cursor Tab Ln 26, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live

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:



Python prime number check

1m · +47 · Auto

Write an efficient Python function that checks whether a number is prime. Use proper variable names and handle edge cases.

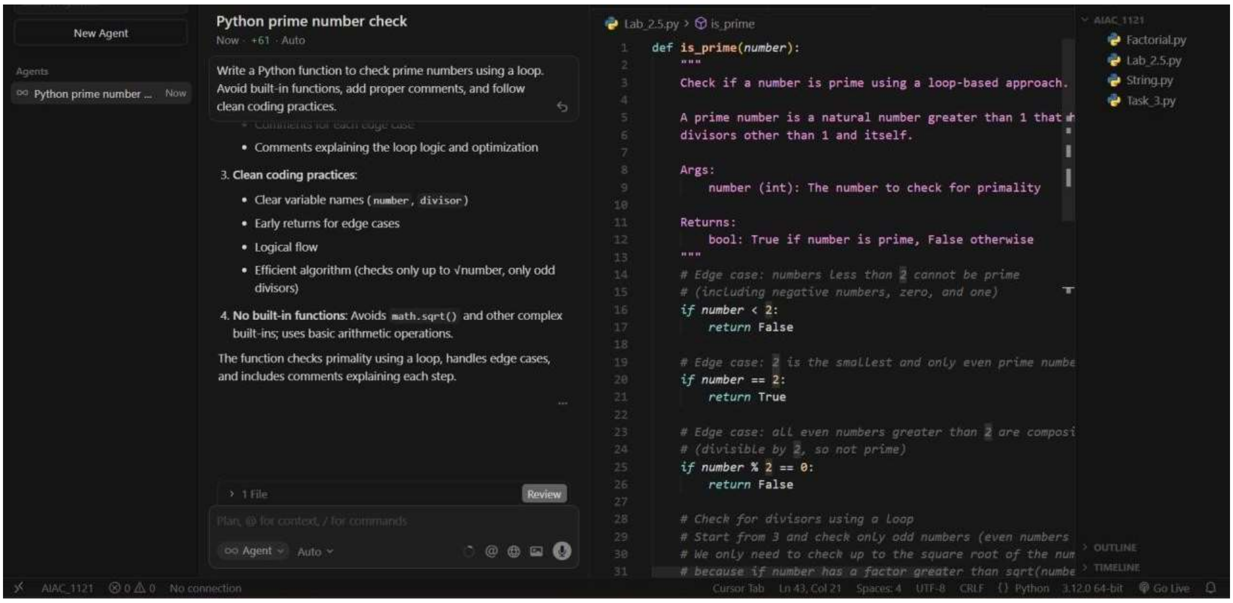
3. **Efficiency:**
   - Checks divisors only up to √number
   - Tests only odd divisors (skips evens after 2)
   - Early returns when a divisor is found
4. **Code quality:**
   - Added docstring
   - Clear comments explaining the logic
   - Proper spacing

The function efficiently checks primality with O(√n) time complexity. It handles all edge cases and uses descriptive variable names.

```python
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
        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}")

```

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

<u>Code:</u>



Python prime number check

Now · +61 · Auto

Write a Python function to check prime numbers using a loop. Avoid built-in functions, add proper comments, and follow clean coding practices.

- Comments for each edge case
- Comments explaining the loop logic and optimization

3. **Clean coding practices:**
   - Clear variable names (`number`, `divisor`)
   - Early returns for edge cases
   - Logical flow
   - Efficient algorithm (checks only up to √number, only odd divisors)
4. **No built-in functions:** Avoids `math.sqrt()` and other complex built-ins; uses basic arithmetic operations.

The function checks primality using a loop, handles edge cases, and includes comments explaining each step.

```python
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
    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 numbe
    if number == 2:
        return True

    # Edge case: all even numbers greater than 2 are composi
    # (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
    # We only need to check up to the square root of the num
    # because if number has a factor greater than sqrt(numbe
```

<u>Task 4:</u> Tool Comparison Reflection.

<u>Prompt:</u> 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 beginnerfriendly, 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.