ASSIGNMENT – 2.5

HT.NO: 2303A52376

BATCH: 45

TASK – 01:

Question:

ask 1: Refactoring Odd/Even Logic (List Version)

❖ Scenario:

You are improving legacy code.

❖ Task:

Write a program to calculate the sum of odd and even numbers in a list,

then refactor it using AI.

Prompt-01:

Write a simple program that takes a list of integers and calculates:

1) The sum of even numbers

2) The sum of odd numbers

Generated Code:

```
def sum_even_odd(numbers):
    sum_even = sum(num for num in numbers if num % 2 == 0)
    sum_odd = sum(num for num in numbers if num % 2 != 0)
    return sum_even, sum_odd


# Example usage
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum, odd_sum = sum_even_odd(numbers)


print(f"Sum of even numbers: {even_sum}")
```

print(f"Sum of odd numbers: {odd_sum}")

Output:

```
PS C:\Users\RIMSHA NAAZ\OneDrive\Documents\AAC-Assignments> & "C:\Users\RIMSHA NAAZ\AppData\L
1\python.exe" "c:/Users/RIMSHA NAAZ/OneDrive/Documents/AAC-Assignments/Assign-02.py"
Sum of even numbers: 30
Sum of odd numbers: 25
PS C:\Users\RIMSHA NAAZ\OneDrive\Documents\AAC-Assignments>
```

Explanation:

The function sum_even_odd(numbers) is defined to take a list of numbers as an argument.

sum_even: It calculates the sum of all even numbers in the list using a generator expression.

sum_odd: It calculates the sum of all odd numbers in the list using another generator expression.

The function returns two values: the sum of even numbers and the sum of odd numbers.

A list of numbers from 1 to 10 is created, and the function is called with this list. The results are printed, showing the sums of even and odd numbers.

Prompt-02:

Refactor the following program using best coding practices.

Improve readability, efficiency, and structure.

Use meaningful variable names and modern programming conventions.

```
def sum_even_odd(numbers):
    sum_even = sum(num for num in numbers if num % 2 == 0)
    sum_odd = sum(num for num in numbers if num % 2 != 0)
    return sum_even, sum_odd


# Example usage
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum, odd_sum = sum_even_odd(numbers)


print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

Explain what improvements were made in the refactored version.

Generated Code:

```
"""Module for calculating sums of even and odd numbers."""


def calculate_even_odd_sums(numbers: list[int]) -> tuple[int, int]:
    """
    Calculate the sum of even and odd numbers in a list.

    Args:
        numbers: List of integers to process.

    Returns:
        Tuple containing (sum_of_even, sum_of_odd).
    """
    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


def main() -> None:
    """Main entry point for the program."""
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    even_sum, odd_sum = calculate_even_odd_sums(numbers)

    print(f"Sum of even numbers: {even_sum}")
    print(f"Sum of odd numbers: {odd_sum}")


if __name__ == "__main__":
```

main()

Output:

```
PS C:\Users\RIMSHA NAAZ\OneDrive\Documents\AAC-Assignments> & "C:\Users\RIMSHA NAAZ\AppData\L
1\python.exe" "c:/Users/RIMSHA NAAZ/OneDrive/Documents/AAC-Assignments/Assign-02.py"
Sum of even numbers: 30
Sum of odd numbers: 25
PS C:\Users\RIMSHA NAAZ\OneDrive\Documents\AAC-Assignments>
```

Explanation:

The code is a module designed to calculate the sums of even and odd numbers from a list of integers.

The function calculate_even_odd_sums takes a list of integers and returns a tuple containing the sum of even numbers and the sum of odd numbers.

The main function initializes a list of numbers from 1 to 10, calls the sum calculation function, and prints the results for both even and odd sums.

The code checks if it is being run as the main program and calls the main function to execute the logic.

TASK-02:

Question:

Task 2: Area Calculation Explanation

❖ Scenario:

You are onboarding a junior developer.

❖ Task:

Ask Gemini to explain a function that calculates the area of different

shapes.

Prompt:

Write a function that calculates the area of different shapes such as:

Circle

Rectangle

Triangle

Generated Code:

```
import math


def area_circle(radius):
    return math.pi * radius ** 2


def area_rectangle(length, width):
    return length * width


def area_triangle(base, height):
    return 0.5 * base * height


# Example usage:
if __name__ == "__main__":
    print("Area of Circle (radius=5):", area_circle(5))
    print("Area of Rectangle (length=4, width=6):", area_rectangle(4, 6))
    print("Area of Triangle (base=3, height=4):", area_triangle(3, 4))
```

Output:

```
AZ\AppData\Local\Programs\Python\Python311\python.exe" "c:/Users/RIMSHA NAAZ/Or
rive/Documents/AAC-Assignments/Assign-02.py"
Area of Circle (radius=5): 78.53981633974483
Area of Rectangle (length=4, width=6): 24
Area of Triangle (base=3, height=4): 6.0
```

Explanation:

The code imports the math module to access the constant pi, which is needed for circle area calculations.

Three functions are defined—area_circle, area_rectangle, and area_triangle—each taking specific parameters and returning the calculated area using their respective geometric formulas.

The if __name__ == "__main__": block ensures the code runs only when executed directly, then calls each area function with example values and prints the results.

The functions use standard formulas: circle ($\pi \times r^2$), rectangle (length × width), and triangle (0.5 × base × height).

TASK-03:

Question:

❖ Scenario:

You are testing how AI responds to different prompts.

❖ Task:

Use Cursor AI with different prompts for the same problem and observe

code changes.

Prompt-01:

Write a program to check whether a number is prime or not.

Generated Code:

```
def is_prime(n):
    """Check whether a number is prime or not."""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

# Get input from user
num = int(input("Enter a number: "))

# Check and display result
if is_prime(num):
```

```
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

Output:

```
PS C:\Users\RIMSHA NAAZ\OneDrive\Documents\AAC-Assignments> & "C:\Users\RIMSHA N
1\python.exe" "c:/Users/RIMSHA NAAZ/OneDrive/Documents/AAC-Assignments/Assign-02
Enter a number: 12
12 is not a prime number.
PS C:\Users\RIMSHA NAAZ\OneDrive\Documents\AAC-Assignments>
```

Explanation:

The is_prime() function determines if a number is prime by first handling special cases (numbers less than 2 are not prime, 2 is prime, and even numbers are not prime).

For other numbers, the function checks if any odd divisor exists from 3 up to the square root of the number; if a divisor is found, the number is not prime.

he program prompts the user to enter a number and stores it in the variable num.

The code calls the is_prime() function and displays whether the entered number is prime or not based on the returned result.

Prompt-02:

Write an optimized program to check whether a number is prime.

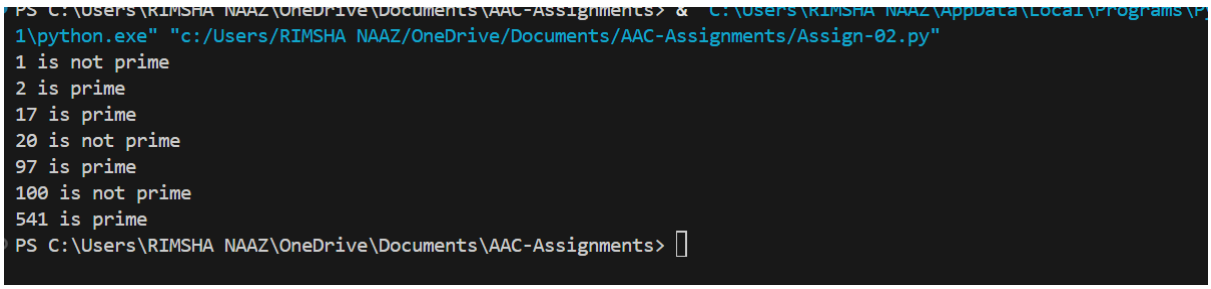
Use efficient logic and minimize time complexity.

Generated Code:

```
def is_prime(n):
    """
    Check if a number is prime with O(√n) time complexity.

    Args:
        n: Integer to check

    Returns:
        Boolean: True if prime, False otherwise
    """
```

```
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    # Check odd divisors up to √n
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2

    return True

# Test cases
if __name__ == "__main__":
    test_numbers = [1, 2, 17, 20, 97, 100, 541]

    for num in test_numbers:
        result = is_prime(num)
        print(f"{num} is {'prime' if result else 'not prime'}")
```

Output:



Explanation:

The is_prime() function efficiently checks if a number is prime using O(√n) time complexity by only checking divisors up to the square root of the number, rather than all numbers up to n.

The function handles special cases—numbers less than 2 return False, the number 2 returns True, and even numbers return False since they're divisible by 2.

Using a while loop, the function checks if any odd divisor exists from 3 up to √n; if found, the number is not prime, otherwise it returns True.

The main block defines a list of test numbers and iterates through each one, calling is_prime() and printing whether each number is prime or not using a conditional expression for concise output.

Prompt-03:

Write a clean, well-structured program to check whether a number is prime.

Use proper function design, comments, and meaningful variable names.

Generated Code:

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

    Args:
        number: An integer to check for primality

    Returns:
        bool: True if the number is prime, False otherwise
    """
    # Handle edge cases
    if number < 2:
        return False

    # 2 is the only even prime number
    if number == 2:
```

```
        return True

    # Exclude other even numbers
    if number % 2 == 0:
        return False

    # Check odd divisors up to the square root of the number
    for divisor in range(3, int(number**0.5) + 1, 2):
        if number % divisor == 0:
            return False

    return True

def main():
    """Main function to run the prime checker program."""
    try:
        user_input = int(input("Enter a number to check if it's prime: "))

        if is_prime(user_input):
            print(f"{user_input} is a prime number.")
        else:
            print(f"{user_input} is not a prime number.")

    except ValueError:
        print("Error: Please enter a valid integer.")

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

Output:

```
rive/Documents/AAC-Assignments/Assign-02.py"
Enter a number to check if it's prime: 17
17 is a prime number.
```

Explanation:

The is_prime() function determines if a number is prime by handling edge cases (numbers less than 2, the number 2, and even numbers) and checking for odd divisors up to the square root of the number.

The main() function takes user input, calls is_prime() to check if the number is prime, displays the result, and uses a try-except block to handle invalid input.

The code runs the main() function when executed directly, allowing users to interactively check if a number is prime.

TASK-04:

Question:

Task 4: Tool Comparison Reflection

❖ Scenario:

You must recommend an AI coding tool.

❖ Task:

Based on your work in this topic, compare Gemini, Copilot, and Cursor AI

for usability and code quality.

Prompt:

Based on practical usage, write a short comparison of the following AI coding tools:

1) Gemini

2) GitHub Copilot

3) Chat GPT

Compare them based on:

- Ease of use

- Code quality

- Explanation clarity

- Suitability for students and professionals

Write the reflection in simple academic language (150–200 words).

Generated Reflection:

Reflection on AI Coding Tools: Gemini, GitHub Copilot, and ChatGPT

In recent practical use, Gemini, GitHub Copilot, and ChatGPT demonstrate distinct strengths across ease of use, code quality, explanation clarity, and suitability for various users. In terms of ease of use, **ChatGPT** and **Gemini** are straightforward because users interact via conversational prompts, which makes them intuitive for beginners. **GitHub Copilot** integrates directly into code editors, which enhances workflow for developers but may require initial setup.

Regarding code quality, **GitHub Copilot** often produces contextually relevant suggestions within development environments, resulting in well-structured code for ongoing projects. **ChatGPT** also generates reliable code, though it sometimes requires refinement when used outside an integrated editor. **Gemini** provides competitive code suggestions with concise patterns, but may be less consistent for complex projects.

In explanation clarity, **ChatGPT** excels by offering detailed, human-like reasoning alongside examples, beneficial for learning. **Gemini** also explains concepts clearly but occasionally with less depth. **GitHub Copilot** typically offers minimal explanations, focusing more on code completion.

For suitability, students benefit most from **ChatGPT** and **Gemini** because of their robust explanations. Professionals with advanced workflows may prefer **GitHub Copilot** for in-editor automation and productivity.

Explanation:

This reflection compares Gemini, GitHub Copilot, and ChatGPT based on how easy they are to use, the quality of code they generate, and how well they explain concepts. It shows that ChatGPT and Gemini are more helpful for students due to clear explanations, while GitHub Copilot is better for professionals who want faster coding inside an editor.