

COURSE: AI Assisted Coding

NAME : G.Sandeep

BATCH-34

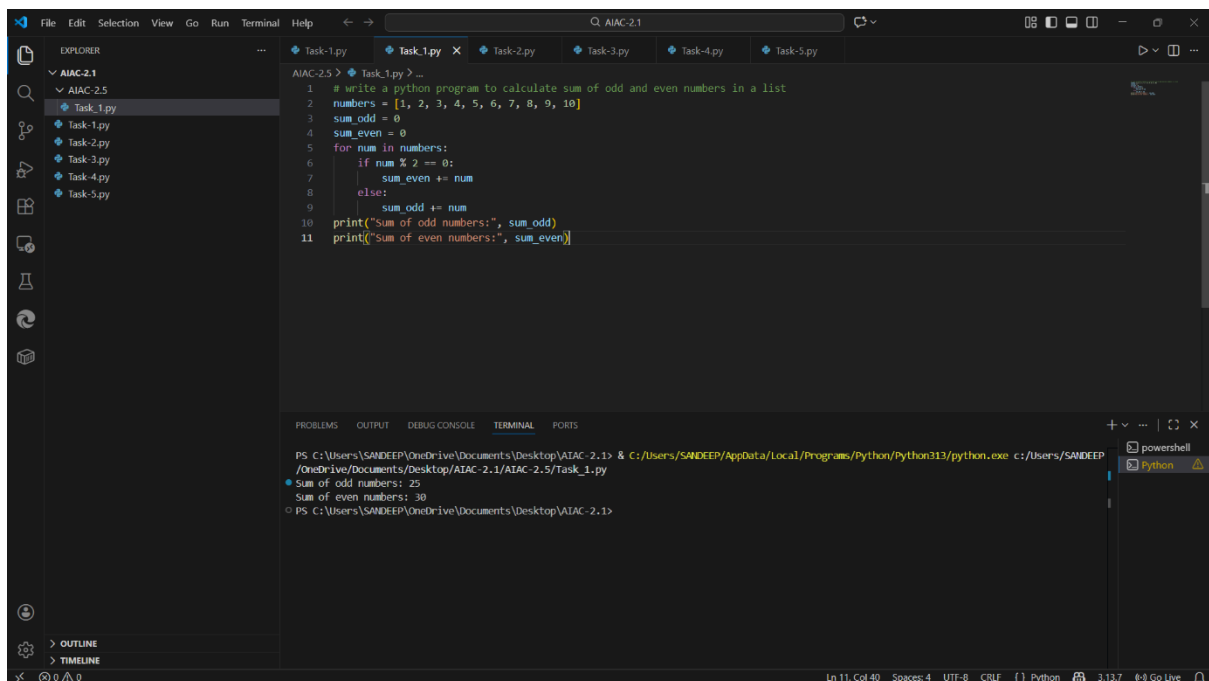
HALLTICKET.NO : 2303A52371

AIAC-ASS-2.5

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

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

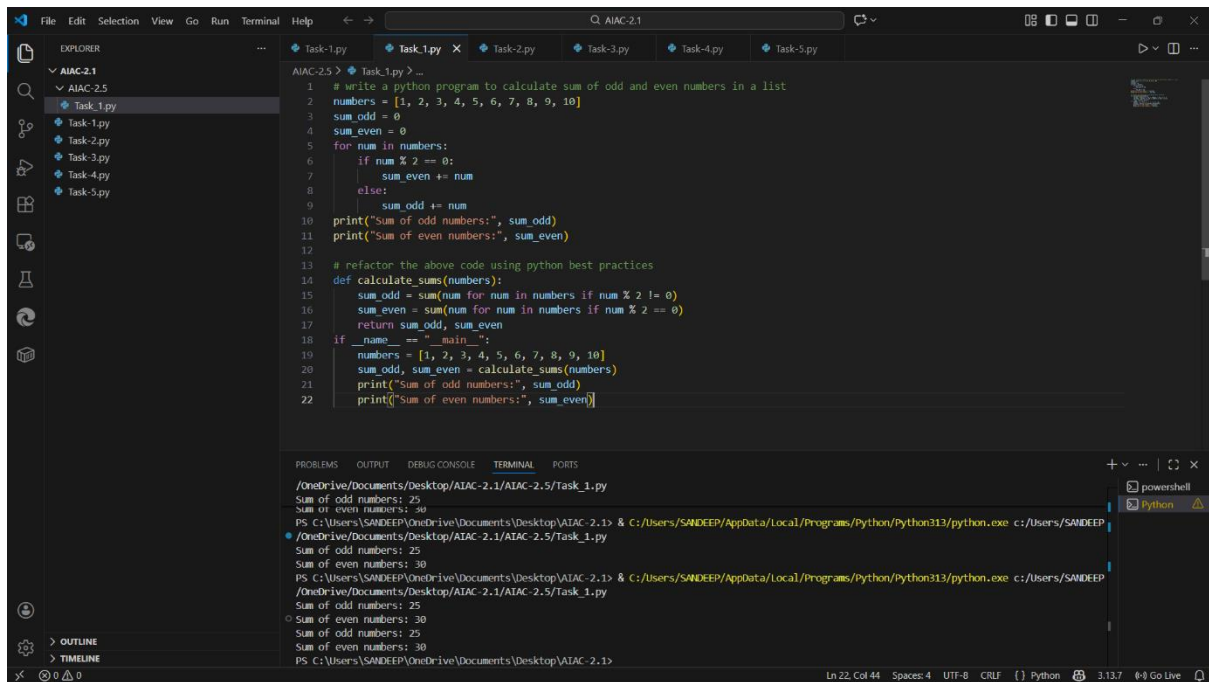
- **Scenario:**
- # write a python program to calculate sum of odd and even numbers in a list



```
AIAC-2.5 > Task_1.py > ...
1 # write a python program to calculate sum of odd and even numbers in a list
2 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3 sum_odd = 0
4 sum_even = 0
5 for num in numbers:
6     if num % 2 == 0:
7         sum_even += num
8     else:
9         sum_odd += num
10 print("Sum of odd numbers:", sum_odd)
11 print("Sum of even numbers:", sum_even)
```

```
PS C:\Users\SANDEEP\OneDrive\Documents\Desktop\AIAC-2.1> & C:\Users\SANDEEP\AppData\Local\Programs\Python\Python313\python.exe c:\Users\SANDEEP\OneDrive\Documents\Desktop\AIAC-2.1\AIAC-2.5\Task_1.py
Sum of odd numbers: 25
Sum of even numbers: 30
PS C:\Users\SANDEEP\OneDrive\Documents\Desktop\AIAC-2.1>
```

refactor the above code using python best practices



The screenshot shows a Visual Studio Code editor with a Python file named `Task_1.py` open. The code defines a list of numbers and calculates the sum of odd and even numbers. The terminal output shows the execution of the script, displaying the sum of odd numbers as 25 and the sum of even numbers as 30.

```
1 # write a python program to calculate sum of odd and even numbers in a list
2 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3 sum_odd = 0
4 sum_even = 0
5 for num in numbers:
6     if num % 2 == 0:
7         sum_even += num
8     else:
9         sum_odd += num
10 print("Sum of odd numbers:", sum_odd)
11 print("Sum of even numbers:", sum_even)
12
13 # refactor the above code using python best practices
14 def calculate_sums(numbers):
15     sum_odd = sum(num for num in numbers if num % 2 != 0)
16     sum_even = sum(num for num in numbers if num % 2 == 0)
17     return sum_odd, sum_even
18 if __name__ == "__main__":
19     numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
20     sum_odd, sum_even = calculate_sums(numbers)
21     print("Sum of odd numbers:", sum_odd)
22     print("Sum of even numbers:", sum_even)
```

Terminal Output:

```
/OneDrive/Documents/Desktop/AIAC-2.1/AIAC-2.5/Task_1.py
Sum of odd numbers: 25
Sum of even numbers: 30
PS C:\Users\SANDEEP\OneDrive\Documents\Desktop\AIAC-2.1> & C:/Users/SANDEEP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/SANDEEP/OneDrive/Documents/Desktop/AIAC-2.1/AIAC-2.5/Task_1.py
Sum of odd numbers: 25
Sum of even numbers: 30
PS C:\Users\SANDEEP\OneDrive\Documents\Desktop\AIAC-2.1> & C:/Users/SANDEEP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/SANDEEP/OneDrive/Documents/Desktop/AIAC-2.1/AIAC-2.5/Task_1.py
Sum of odd numbers: 25
Sum of even numbers: 30
PS C:\Users\SANDEEP\OneDrive\Documents\Desktop\AIAC-2.1>
```

Task 2: Area Calculation Explanation

Scenario

You are onboarding a junior developer and need to explain how a function calculates the area of different shapes.

❖ Task:

Prompt Used in Google Gemini (Colab)

“Explain a Python function that calculates the area of different shapes like circle, rectangle, and triangle.”

❖ Expected Output:

➤ Code

The image shows a Google Colab notebook titled "AIAC-ASS-2.5.ipynb". The code defines a function `calculate_area(shape_type, **kwargs)` that calculates the area of different geometric shapes. The function uses conditional logic to handle circles, rectangles, and triangles, with appropriate error messages for invalid inputs. The output shows the area calculations for a circle, rectangle, and triangle, along with error messages for invalid shape types and missing dimensions.

```
def calculate_area(shape_type, **kwargs):
    """
    Calculates the area of different geometric shapes.

    Args:
        shape_type (str): The type of the shape ('circle', 'rectangle', 'triangle').
        **kwargs: Keyword arguments for shape-specific dimensions:
            - For 'circle': 'radius' (float)
            - For 'rectangle': 'length' (float), 'width' (float)
            - For 'triangle': 'base' (float), 'height' (float)

    Returns:
        float: The calculated area of the shape.
        str: An error message if the shape type is invalid or required dimensions are missing.
    """
    shape_type = shape_type.lower()

    if shape_type == 'circle':
        if 'radius' in kwargs:
            radius = kwargs['radius']
            return math.pi * (radius ** 2)
        else:
            return "Error: Radius is required for a circle."
    elif shape_type == 'rectangle':
        if 'length' in kwargs and 'width' in kwargs:
            length = kwargs['length']
            width = kwargs['width']
            return length * width
        else:
            return "Error: Length and width are required for a rectangle."
    elif shape_type == 'triangle':
        if 'base' in kwargs and 'height' in kwargs:
            base = kwargs['base']
            height = kwargs['height']
            return 0.5 * base * height
        else:
            return "Error: Base and height are required for a triangle."
    else:
        return "Error: Invalid shape type. Choose 'circle', 'rectangle', or 'triangle'."

# Example Usage:
print(f"Area of circle with radius 5: {calculate_area('circle', radius=5)}")
print(f"Area of rectangle with length 4 and width 6: {calculate_area('rectangle', length=4, width=6)}")
print(f"Area of triangle with base 7 and height 3: {calculate_area('triangle', base=7, height=3)}")
print(f"Invalid shape: {calculate_area('square', side=5)}")
print(f"Missing dimension for circle: {calculate_area('circle')}")

"""
Area of circle with radius 5: 78.53981633974483
Area of rectangle with length 4 and width 6: 24
Area of triangle with base 7 and height 3: 10.5
Invalid shape: Error: Invalid shape type. Choose 'circle', 'rectangle', or 'triangle'.
Missing dimension for circle: Error: Radius is required for a circle.
"""
```

The image shows a Google Colab notebook titled "AIAC-ASS-2.5.ipynb". The code provides a detailed explanation of the `calculate_area` function, covering its imports, arguments, logic, and error handling.

```
Explanation of the calculate_area function:

1. import math: This line imports the 'math' module, which is necessary to use 'math.pi' for calculating the area of a circle.

2. def calculate_area(shape_type, **kwargs):
    - This defines a function named 'calculate_area'.
    - 'shape_type' is a positional argument that expects a string (e.g., 'circle', 'rectangle', 'triangle') to specify the shape.
    - '**kwargs' (keyword arguments) allows the function to accept an arbitrary number of keyword arguments. This is useful because different shapes require different parameters (e.g., 'radius' for a circle, 'length' and 'width' for a rectangle).

3. shape_type = shape_type.lower():
    - Converts the input 'shape_type' to lowercase to make the function case-insensitive (e.g., 'Circle' will be treated the same as 'circle').

4. Conditional Logic (if-elif-else):
    - The function uses 'if-elif-else' statements to check the value of 'shape_type' and execute the appropriate area calculation logic.

5. For 'circle':
    - It checks if a 'radius' keyword argument was provided in 'kwargs'.
    - If 'radius' is present, it calculates the area using the formula:  $\pi \times \text{radius}^2$ .
    - If 'radius' is missing, it returns an error message.

6. For 'rectangle':
    - It checks if both 'length' and 'width' keyword arguments were provided.
    - If both are present, it calculates the area using the formula:  $\text{length} \times \text{width}$ .
    - If either is missing, it returns an error message.

7. For 'triangle':
    - It checks if both 'base' and 'height' keyword arguments were provided.
    - If both are present, it calculates the area using the formula:  $0.5 \times \text{base} \times \text{height}$ .
    - If either is missing, it returns an error message.

8. else block:
    - If 'shape_type' does not match any of the recognized shapes, it returns an
```

Explanation

- The function `calculate_area()` is used to calculate the area of various shapes.
- The parameter `shape` specifies the type of shape (circle, rectangle, or triangle).
- `value1` and `value2` represent the required dimensions of the shape.
- For a **circle**, the area is calculated using the formula $\pi \times \text{radius}^2$.
- For a **rectangle**, the area is calculated as $\text{length} \times \text{breadth}$.

- For a **triangle**, the area is calculated as $\frac{1}{2} \times \text{base} \times \text{height}$.
- If an unknown shape is passed, the function returns "Invalid shape".

Task 3: Prompt Sensitivity Experiment

Scenario

You are testing how Cursor AI responds to different prompts for the same problem and observing how the generated code changes.

Problem Selected

Calculate the factorial of a number.

Prompt List and Code Variations (Using Cursor AI)

Prompt 1

Prompts:

#Write a Python program to calculate factorial of a number.”

#Create a Python program to find factorial iteratively.

#Create a Python program to find factorial using math module.

#Create a Python program to find factorial using reduce function from functools module.

The screenshot shows the Cursor AI IDE interface. The main editor displays a Python file named 'factorial.py' with the following code:

```

1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n - 1)
6
7 # Example usage
8 print("Factorial of 5 is:", factorial(5)) # Output: 120
9
10
11 #Create a Python program to find factorial iteratively.
12 def factorial_iterative(n):
13     result = 1
14     for i in range(1, n + 1):
15         result *= i
16     return result
17
18 print("Factorial of 5 is:", factorial_iterative(5)) # Output: 120
19
20 #Create a Python program to find factorial using math module.
21 import math
22 print("Factorial of 5 is:", math.factorial(5)) # Output: 120
23
24 #Create a Python program to find factorial using reduce function from functools module.
25 from functools import reduce
26 def factorial_reduce(n):
27     return reduce(lambda x, y: x * y, range(1, n + 1), 1)
28 print("Factorial of 5 is:", factorial_reduce(5)) # Output: 120
29

```

The terminal output at the bottom shows the execution of the program, displaying the factorial of 5 as 120 for all four methods.

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.

❖ Expected Output:

Short written reflection

Tool Comparison Reflection

During this experiment with AI-assisted coding, I explored three major AI coding tools: **Gemini**, **GitHub Copilot**, and **Cursor AI**. Each tool has strengths and limitations in terms of **usability** and **code quality**.

1. Gemini

- **Usability:** Gemini provides a clear interface for generating code and explanations. It's beginner-friendly, and the AI can respond to natural language prompts directly.
- **Code Quality:** Gemini often generates readable code with proper comments and stepwise logic. However, sometimes it includes extra steps that may not be necessary, slightly reducing efficiency.
- **Best Use Case:** Learning and understanding code logic, especially for beginners.

2. GitHub Copilot

- **Usability:** Copilot integrates seamlessly into VS Code. Code suggestions appear inline, making coding faster. Accepting or cycling through suggestions is intuitive.
- **Code Quality:** Copilot usually produces functional code that follows common programming practices. It is strong for generating boilerplate code and common algorithms. However, complex or niche problems may require manual adjustments.
- **Best Use Case:** Professional coding and rapid prototyping in known programming languages.

3. Cursor AI

- **Usability:** Cursor AI allows prompt-based code generation and editing directly in VS Code. It supports various ways to give instructions, like comments or selected code blocks, making it flexible.
- **Code Quality:** Cursor AI generates concise and efficient code, often with multiple variations based on prompts. It is particularly useful for experimenting with different coding approaches and learning alternative methods.
- **Best Use Case:** Testing code variations, experimenting with different programming approaches, and learning multiple solutions for the same problem.

Recommendation:

Based on this experience, **Cursor AI** is the most versatile tool for experimentation and learning because it provides multiple code variations and responds well to diverse prompts. **GitHub Copilot** is best for rapid coding and industry-level projects, while **Gemini** excels in educational scenarios and step-by-step code explanation.