AI ASSISTED CODING

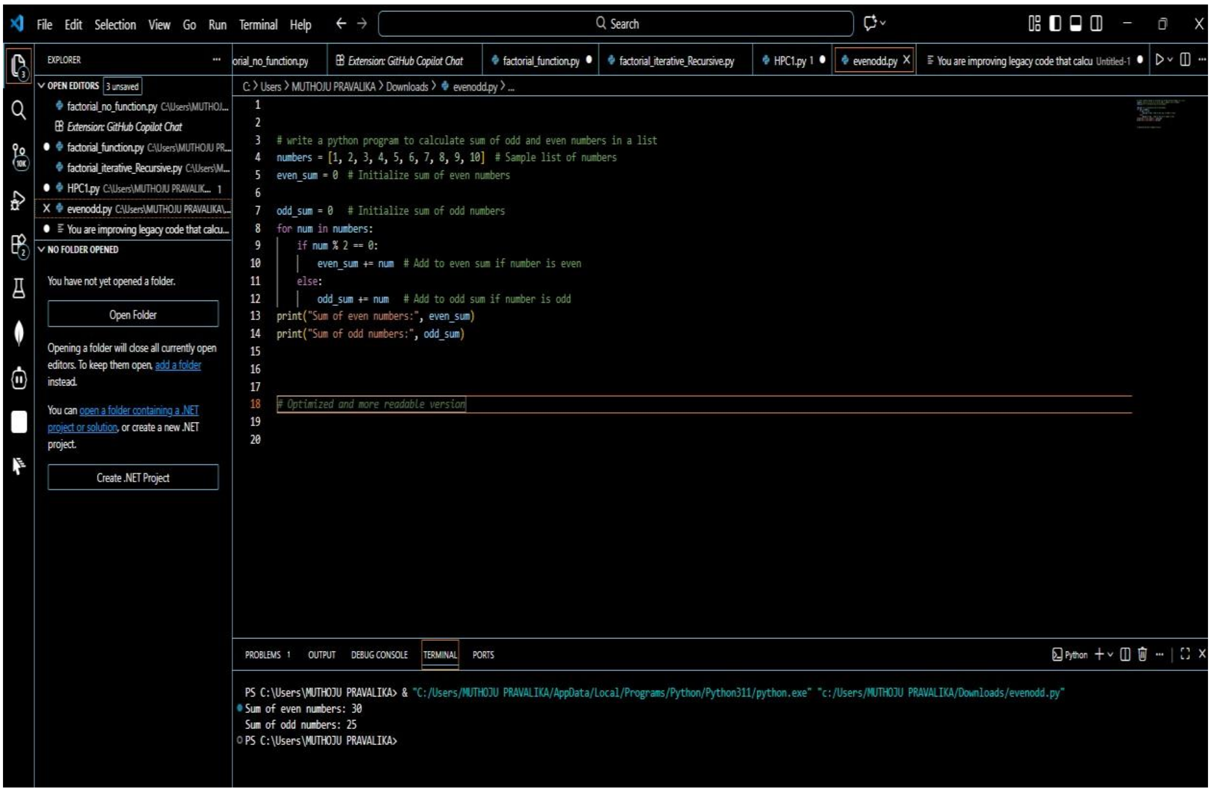
2303A52454

B-34

LAB-2:

Scenario:

write a python program to calculate sum of odd and even numbers in a list

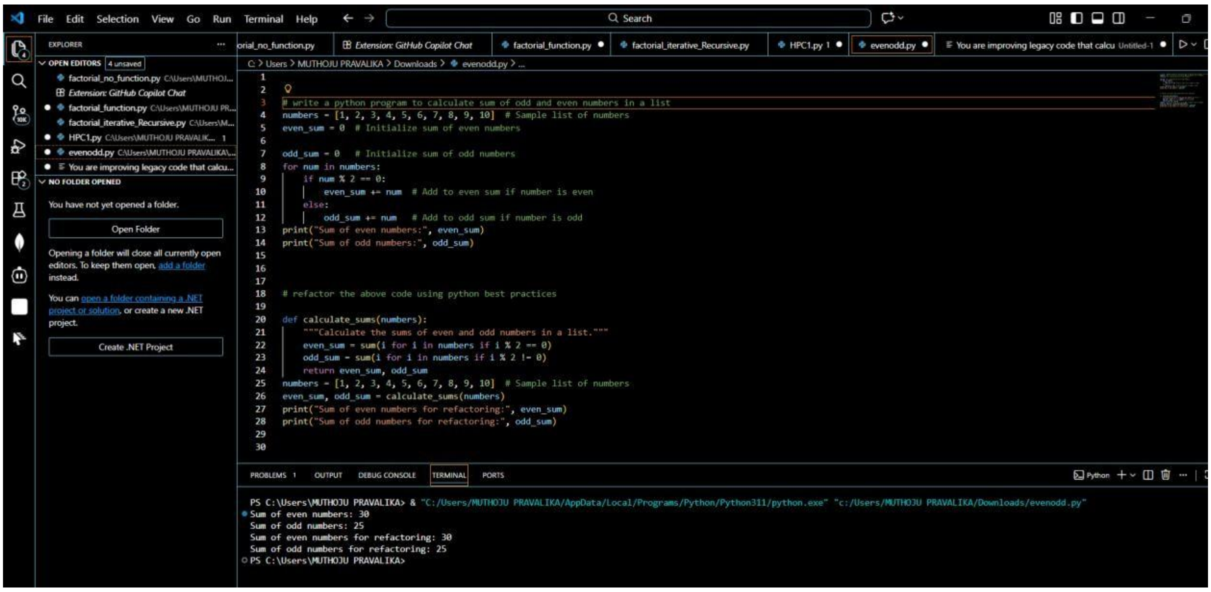


File Edit Selection View Go Run Terminal Help
Search
EXPLORER
OPEN EDITORS 3 unsaved
factorial_no_function.py
Extension: GitHub Copilot Chat
factorial_function.py
factorial_iterative_Recursive.py
HPC1.py
evenodd.py
You are improving legacy code that calcu...
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all currently open editors. To keep them open, add a folder instead.
You can open a folder containing a .NET project or solution, or create a new .NET project.
Create .NET Project
C: > Users > MUTHOJU PRAVALIKA > Downloads > evenodd.py > ...
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python

```python


# write a python program to calculate sum of odd and even numbers in a list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # Sample list of numbers
even_sum = 0  # Initialize sum of even numbers

odd_sum = 0   # Initialize sum of odd numbers
for num in numbers:
    if num % 2 == 0:
        even_sum += num  # Add to even sum if number is even
    else:
        odd_sum += num   # Add to odd sum if number is odd
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)



# Optimized and more readable version


```

```
PS C:\Users\MUTHOJU PRAVALIKA> & "C:/Users/MUTHOJU PRAVALIKA/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/MUTHOJU PRAVALIKA/Downloads/evenodd.py"
Sum of even numbers: 30
Sum of odd numbers: 25
PS C:\Users\MUTHOJU PRAVALIKA>
```

refactor the above code using python best practices



```python


# write a python program to calculate sum of odd and even numbers in a list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # Sample list of numbers
even_sum = 0  # Initialize sum of even numbers

odd_sum = 0   # Initialize sum of odd numbers
for num in numbers:
    if num % 2 == 0:
        even_sum += num  # Add to even sum if number is even
    else:
        odd_sum += num   # Add to odd sum if number is odd
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)



# refactor the above code using python best practices

def calculate_sums(numbers):
    """Calculate the sums of even and odd numbers in a list."""
    even_sum = sum(i for i in numbers if i % 2 == 0)
    odd_sum = sum(i for i in numbers if i % 2 != 0)
    return even_sum, odd_sum
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # Sample list of numbers
even_sum, odd_sum = calculate_sums(numbers)
print("Sum of even numbers for refactoring:", even_sum)
print("Sum of odd numbers for refactoring:", odd_sum)
```

```
PS C:\Users\MUTHOJU PRAVALIKA> & "C:/Users/MUTHOJU PRAVALIKA/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/MUTHOJU PRAVALIKA/Downloads/evenodd.py"
Sum of even numbers: 30
Sum of odd numbers: 25
Sum of even numbers for refactoring: 30
Sum of odd numbers for refactoring: 25
PS C:\Users\MUTHOJU PRAVALIKA>
```

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.

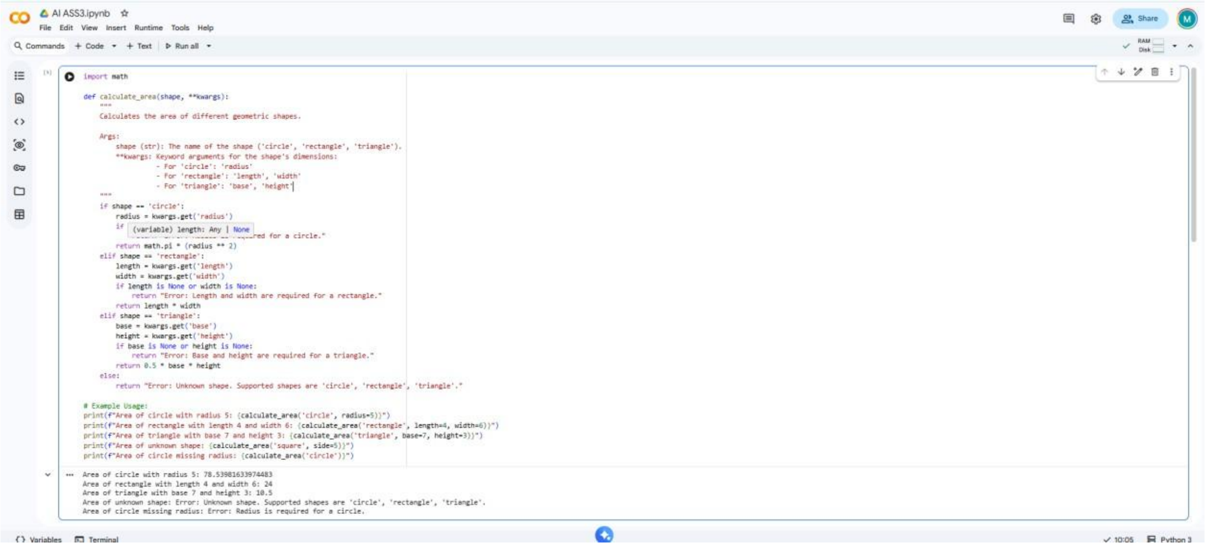
❖ Task:

Prompt Used in Google Gemini (Colab)

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

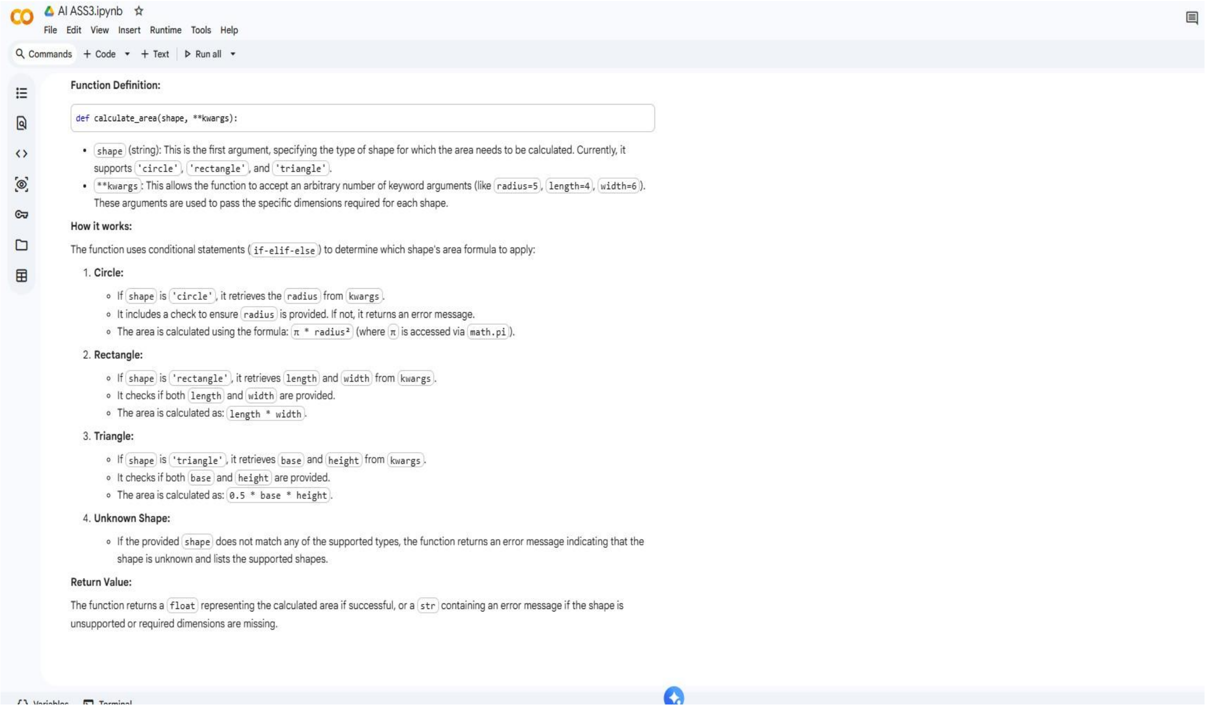
❖ Expected Output:

➢ Code


AI ASS3.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
Variables Terminal
10:05 Python 3

```
import math

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

    Args:
        shape (str): The name of the shape ('circle', 'rectangle', 'triangle').
        **kwargs: Keyword arguments for the shape's dimensions:
                  - For 'circle': 'radius'
                  - For 'rectangle': 'length', 'width'
                  - For 'triangle': 'base', 'height'
    """
    if shape == 'circle':
        radius = kwargs.get('radius')
        if  (variable) length: Any | None
                                 red for a circle."
        return math.pi * (radius ** 2)
    elif shape == 'rectangle':
        length = kwargs.get('length')
        width = kwargs.get('width')
        if length is None or width is None:
            return "Error: Length and width are required for a rectangle."
        return length * width
    elif shape == 'triangle':
        base = kwargs.get('base')
        height = kwargs.get('height')
        if base is None or height is None:
            return "Error: Base and height are required for a triangle."
        return 0.5 * base * height
    else:
        return "Error: Unknown shape. Supported shapes are 'circle', 'rectangle', '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"Area of unknown shape: {calculate_area('square', side=5)}")
print(f"Area of circle missing radius: {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
Area of unknown shape: Error: Unknown shape. Supported shapes are 'circle', 'rectangle', 'triangle'.
Area of circle missing radius: Error: Radius is required for a circle.
```


AI ASS3.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Variables Terminal

Function Definition:

```
def calculate_area(shape, **kwargs):
```

- `shape` (string): This is the first argument, specifying the type of shape for which the area needs to be calculated. Currently, it supports `'circle'`, `'rectangle'`, and `'triangle'`.
- `**kwargs`: This allows the function to accept an arbitrary number of keyword arguments (like `radius=5`, `length=4`, `width=6`). These arguments are used to pass the specific dimensions required for each shape.

How it works:

The function uses conditional statements (`if-elif-else`) to determine which shape's area formula to apply:

1. **Circle:**
   - If `shape` is `'circle'`, it retrieves the `radius` from `kwargs`.
   - It includes a check to ensure `radius` is provided. If not, it returns an error message.
   - The area is calculated using the formula: `π * radius²` (where `π` is accessed via `math.pi`).
2. **Rectangle:**
   - If `shape` is `'rectangle'`, it retrieves `length` and `width` from `kwargs`.
   - It checks if both `length` and `width` are provided.
   - The area is calculated as: `length * width`.
3. **Triangle:**
   - If `shape` is `'triangle'`, it retrieves `base` and `height` from `kwargs`.
   - It checks if both `base` and `height` are provided.
   - The area is calculated as: `0.5 * base * height`.
4. **Unknown Shape:**
   - If the provided `shape` does not match any of the supported types, the function returns an error message indicating that the shape is unknown and lists the supported shapes.

Return Value:

The function returns a `float` representing the calculated area if successful, or a `str` containing an error message if the shape is unsupported or required dimensions are missing.

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 π × radius².
- For a **rectangle**, the area is calculated as length × breadth.
- For a **triangle**, the area is calculated as ½ × base × 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

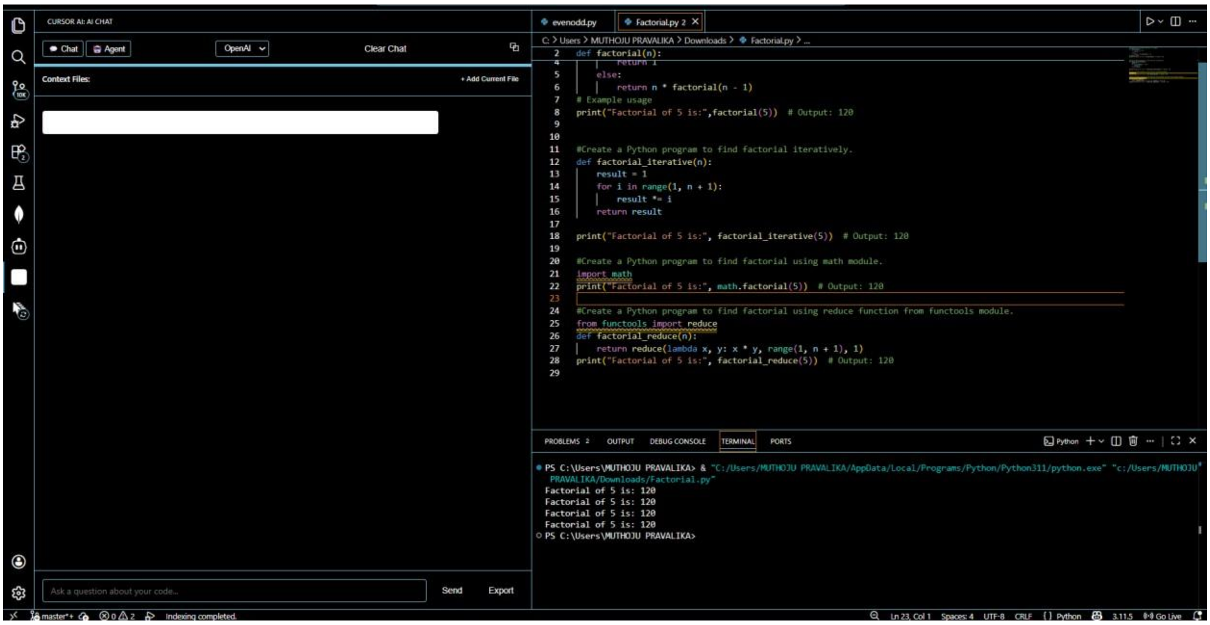
Prompts:

#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.



```python
def factorial(n):
        return 1
    else:
        return n * factorial(n - 1)
# Example usage
print("Factorial of 5 is:",factorial(5))  # Output: 120


#Create a Python program to find factorial iteratively.
def factorial_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

print("Factorial of 5 is:", factorial_iterative(5))  # Output: 120

#Create a Python program to find factorial using math module.
import math
print("Factorial of 5 is:", math.factorial(5))  # Output: 120

#Create a Python program to find factorial using reduce function from functools module.
from functools import reduce
def factorial_reduce(n):
    return reduce(lambda x, y: x * y, range(1, n + 1), 1)
print("Factorial of 5 is:", factorial_reduce(5))  # Output: 120
```

```
PS C:\Users\MUTHOJU PRAVALIKA> & "C:/Users/MUTHOJU PRAVALIKA/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/MUTHOJU
 PRAVALIKA/Downloads/Factorial.py"
Factorial of 5 is: 120
Factorial of 5 is: 120
Factorial of 5 is: 120
Factorial of 5 is: 120
PS C:\Users\MUTHOJU PRAVALIKA>
```

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.