

COURSE: AI Assisted Coding

NAME : PRAVALIKA MUTHOJU

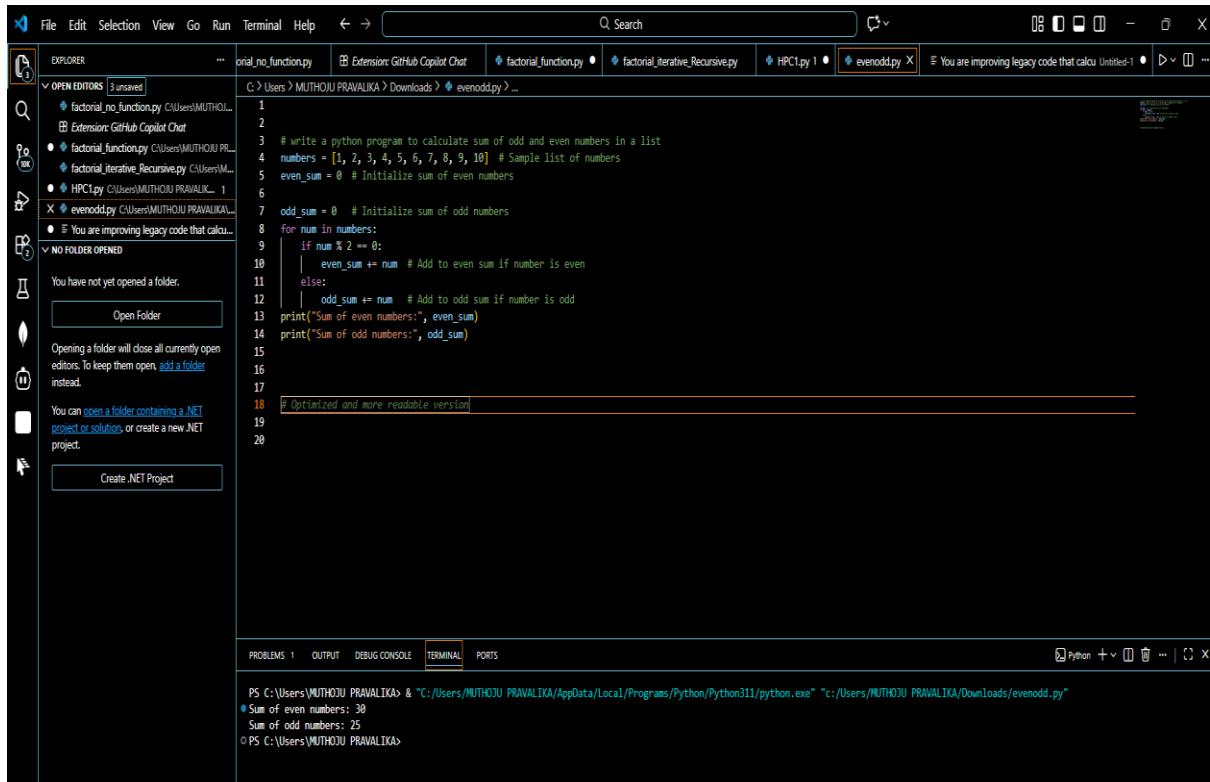
BATCH-34

HALLTICKET.NO : 2303A52347

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



The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar has sections for Explorer, Open Editors, and NO FOLDER OPENED. The main area displays a Python script named evenodd.py with the following code:

```
1 # write a python program to calculate sum of odd and even numbers in a list
2
3 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # Sample list of numbers
4 even_sum = 0 # Initialize sum of even numbers
5 odd_sum = 0 # Initialize sum of odd numbers
6
7 for num in numbers:
8     if num % 2 == 0:
9         even_sum += num # Add to even sum if number is even
10    else:
11        odd_sum += num # Add to odd sum if number is odd
12
13 print("Sum of even numbers:", even_sum)
14 print("Sum of odd numbers:", odd_sum)
15
16
17 # Optimized and more readable version
18
19
20
```

The bottom right corner shows a terminal window with the following output:

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

```
# refactor the above code using python best practices
```

```
File Edit Selection View Go Run Terminal Help < > Q Search

EXPLORER ... orial_no_function.py Extension: GitHub Copilot Chat factorial_function.py factorial_iterative_Recursive.py HPC1.py 1 evenodd.py You are improving legacy code that calcu Untitled-1

C:\Users\UMUTHOUJU PRAVALIKA\Downloads>evenodd.py ...

1
2
3 # write a python program to calculate sum of odd and even numbers in a list
4 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # Sample list of numbers
5 even_sum = 0 # Initialize sum of even numbers
6
7 odd_sum = 0 # Initialize sum of odd numbers
8 for num in numbers:
9     if num % 2 == 0:
10         even_sum += num # Add to even sum if number is even
11     else:
12         odd_sum += num # Add to odd sum if number is odd
13 print("Sum of even numbers:", even_sum)
14 print("Sum of odd numbers:", odd_sum)
15
16
17 # refactor the above code using python best practices
18
19
20 def calculate_sum(numbers):
21     """Calculate the sum of even and odd numbers in a list."""
22     even_sum = sum([i for i in numbers if i % 2 == 0])
23     odd_sum = sum([i for i in numbers if i % 2 != 0])
24     return even_sum, odd_sum
25
26 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # Sample list of numbers
27 even_sum, odd_sum = calculate_sum(numbers)
28 print("Sum of even numbers for refactoring:", even_sum)
29 print("Sum of odd numbers for refactoring:", odd_sum)
30

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\UMUTHOUJU PRAVALIKA> "C:/Users/UMUTHOUJU PRAVALIKA/AppData/Local/Programs/Python/Python311/python.exe" "C:/Users/UMUTHOUJU 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\UMUTHOUJU 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

The screenshot shows two Jupyter Notebook cells. The top cell contains the Python code for the `calculate_area` function, which calculates the area of different geometric shapes based on their type and dimensions. The bottom cell displays the generated documentation for the same function, detailing its parameters, how it works, and return values.

```

import math

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

    Args:
        shape (str): The type 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'

    Returns:
        float: The calculated area.
        str: An error message if the shape is unknown or dimensions are missing.

    Example Usage:
    print("Area of circle with radius 5: ", calculate_area('circle', radius=5))
    print("Area of rectangle with length 4 and width 6: ", calculate_area('rectangle', length=4, width=6))
    print("Area of triangle with base 7 and height 3: ", calculate_area('triangle', base=7, height=3))
    print("Area of unknown shape: ", calculate_area('square', side=5))
    print("Area of circle missing radius: ", calculate_area('circle'))
    """

    if shape == 'circle':
        radius = kwargs.get('radius')
        if ((variable) length: Any | None):
            return math.pi * (radius ** 2)
        else:
            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("Area of circle with radius 5: ", calculate_area('circle', radius=5))
print("Area of rectangle with length 4 and width 6: ", calculate_area('rectangle', length=4, width=6))
print("Area of triangle with base 7 and height 3: ", calculate_area('triangle', base=7, height=3))
print("Area of unknown shape: ", calculate_area('square', side=5))
print("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.

```

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: `$\pi \times radius^2$` (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 $\pi \times radius^2$.
- For a **rectangle**, the area is calculated as $length \times breadth$.
- For a **triangle**, the area is calculated as $\frac{1}{2} \times base \times height$.
- 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 interface with four code snippets in a code editor:

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

Below the code editor is a terminal window showing the execution of the code:

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

At the bottom, there is a status bar with information like In:23 Col:1 Spaces:4 UFT-8 CR/LF [Python] 331.5 88% Online.

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.