

# AI Assisted Coding

## Assignment 2.5

Name: Ch. Koushik

Hall ticket no: 2303A51938

Batch no: 19

### Task 1:

#### Prompt:

Write a program to calculate the sum of odd and even numbers in a list, then improve it using AI.

#### Code & Output:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Assignment Number-2.5.ipynb
- Code Cell Content:**

```
[10]: 1 my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 odd_sum = 0
4 even_sum = 0
5
6 for number in my_list:
7     if number % 2 == 0:
8         even_sum += number
9     else:
10        odd_sum += number
11
12 print(f"Original Code Output:")
13 print(f"Sum of odd numbers: {odd_sum}")
14 print(f"Sum of even numbers: {even_sum}")
```
- Output Cell Content:**

```
... Original Code Output:
Sum of odd numbers: 25
Sum of even numbers: 30
```
- Bottom Status Bar:** 8:37PM Python 3

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Assignment Number-2.5.ipynb
- Code Cell Content:**

```
[11]: ... Sum of even numbers: 30
```
- Code Cell Content:**

```
[11]: 1 my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 # Using list comprehension and sum() for a more concise solution
4 odd_sum_improved = sum(number for number in my_list if number % 2 != 0)
5 even_sum_improved = sum(number for number in my_list if number % 2 == 0)
6
7 print(f"\nImproved Code Output:")
8 print(f"Sum of odd numbers (improved): {odd_sum_improved}")
9 print(f"Sum of even numbers (improved): {even_sum_improved}")
```
- Output Cell Content:**

```
Improved Code Output:
Sum of odd numbers (improved): 25
Sum of even numbers (improved): 30
```
- Bottom Status Bar:** 8:37PM Python 3

### **Explanation:**

The original code uses a traditional for loop with if/else conditions to iterate through the list and accumulate sums. While functional, it is more verbose.

The improved code leverages Python's built-in functionalities:

- **List Comprehension (Generator Expression):** Instead of creating intermediate lists, a generator expression (number for number in my\_list if condition) is used directly within sum(). This is memory-efficient as it generates values on the fly.
- **sum() function:** The sum() function is a highly optimized built-in that efficiently calculates the total of all items in an iterable.

This refactored approach is more *Pythonic*, concise, and often more readable for experienced Python developers, especially for simpler aggregation tasks.

### **Task 2:**

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

#### **Prompt:**

Explain a function that calculates the area of different shapes.

#### **Code:**

```
import math
```

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

```
    """
```

Calculates the area of different geometric shapes.

Args:

shape (str): The type of shape (e.g., 'circle', 'rectangle', 'triangle').

\*\*kwargs: Keyword arguments for shape dimensions:

- 'radius' for circle
- 'length', 'width' for rectangle
- 'base', 'height' for triangle

Returns:

float: The calculated area of the shape.

str: An error message if the shape is unknown or dimensions are missing.

```
    """
```

```
if shape.lower() == 'circle':
```

```
    radius = kwargs.get('radius')
```

```
if radius is not None:
    return math.pi * (radius ** 2)
else:
    return "Error: Circle requires 'radius'!"

elif shape.lower() == 'rectangle':
    length = kwargs.get('length')
    width = kwargs.get('width')
    if length is not None and width is not None:
        return length * width
    else:
        return "Error: Rectangle requires 'length' and 'width'!"

elif shape.lower() == 'triangle':
    base = kwargs.get('base')
    height = kwargs.get('height')
    if base is not None and height is not None:
        return 0.5 * base * height
    else:
        return "Error: Triangle requires 'base' and 'height'!"

else:
    return f"Error: Unknown shape '{shape}'"

# --- Example Usage ---

print("--- Area Calculations ---")

# Circle
circle_area = calculate_area('circle', radius=5)
print(f"Area of a circle with radius 5: {circle_area:.2f}")

# Rectangle
rectangle_area = calculate_area('rectangle', length=4, width=6)
print(f"Area of a rectangle with length 4 and width 6: {rectangle_area}")

# Triangle
triangle_area = calculate_area('triangle', base=10, height=3)
print(f"Area of a triangle with base 10 and height 3: {triangle_area}")
```

```

# Invalid shape
invalid_shape_area = calculate_area('square', side=5)
print(f"Invalid shape calculation: {invalid_shape_area}")

# Missing dimensions
missing_dim_area = calculate_area('circle')
print(f"Missing dimension calculation: {missing_dim_area}")

```

### **Output:**

--- Area Calculations ---

Area of a circle with radius 5: 78.54

Area of a rectangle with length 4 and width 6: 24

Area of a triangle with base 10 and height 3: 15.0

Invalid shape calculation: Error: Unknown shape 'square'.

Missing dimension calculation: Error: Circle requires 'radius'.

### **Explanation:**

This `calculate_area` function is a versatile tool for finding the area of different shapes like circles, rectangles, and triangles. It uses a single function to handle multiple tasks, which is a good practice for organizing code.

#### Key Ideas:

1. **import math:** We import the math library to use `math.pi` (the value of Pi) for circle calculations.
2. **def calculate\_area(shape, \*\*kwargs):**: This defines the function:
  - o **shape:** This tells the function *which* shape to calculate (e.g., 'circle').
  - o **\*\*kwargs:** This is a flexible way to pass *extra information* like `radius=5`, `length=4`, etc. It gathers these into a dictionary.
3. **if/elif/else for Different Shapes:** The function checks the shape you provide:
  - o If it's a **circle**, it looks for radius and uses  $\pi * \text{radius}^2$ .
  - o If it's a **rectangle**, it looks for length and width and uses `length * width`.
  - o If it's a **triangle**, it looks for base and height and uses  $0.5 * \text{base} * \text{height}$ .
4. **Error Handling:** If you forget to provide a necessary dimension (like radius for a circle) or provide an unknown shape, the function returns a helpful error message instead of crashing. This makes the function user-friendly.

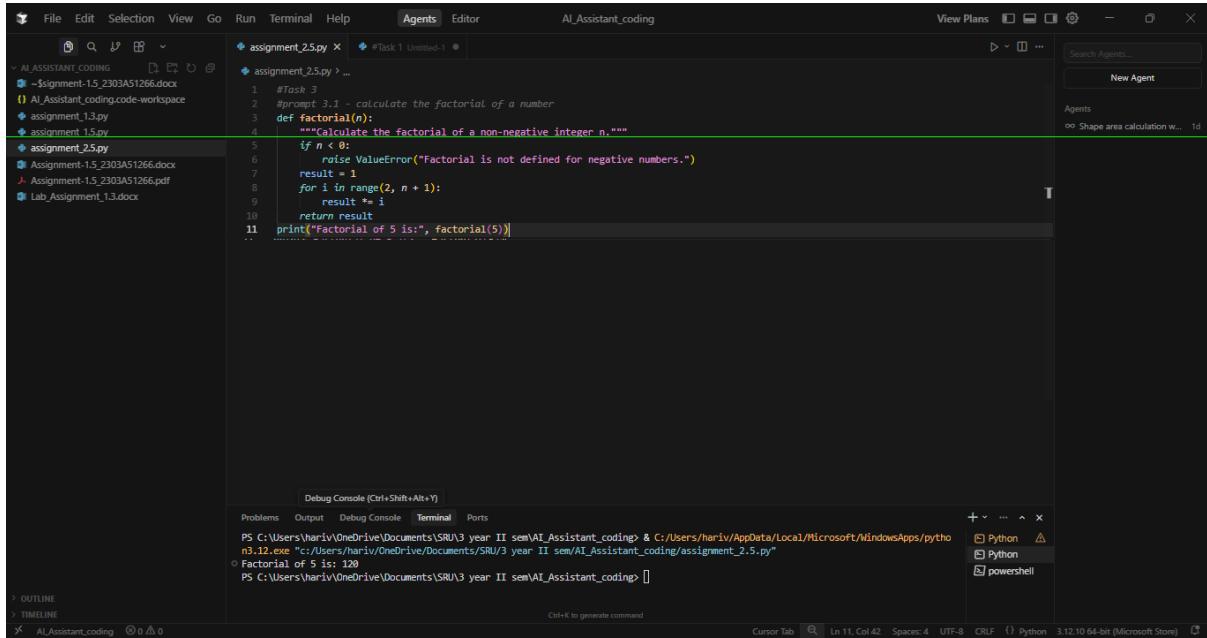
### Task 3:

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

#### Prompt 3.1:

Calculate the factorial of a number

#### Code & Output:



The screenshot shows the Microsoft Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, Agents, Editor, and AI\_Assistant\_coding. The left sidebar shows a file tree with files like assignment\_1.3.py, assignment\_1.5.py, assignment\_2.5.py, and assignment\_2.5.py (the active tab). The main editor area contains the following Python code:

```
#Task 3
#prompt: 3.1 - calculate the factorial of a number
def factorial(n):
    """Calculate the factorial of a non-negative integer n."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
print("Factorial of 5 is:", factorial(5))
```

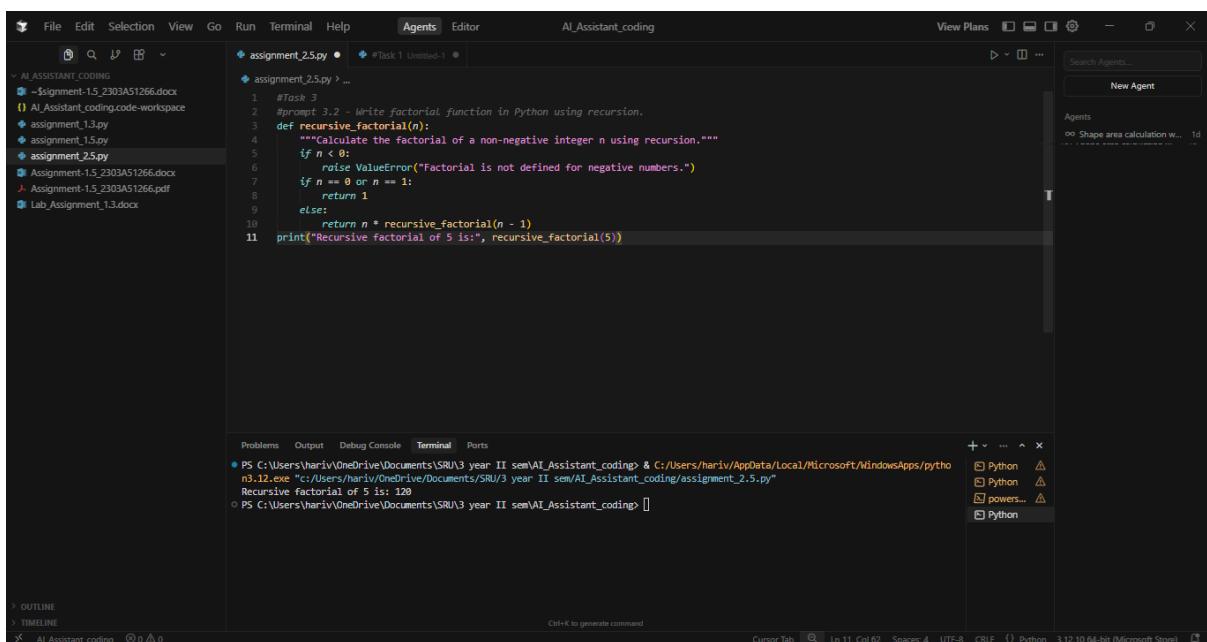
The terminal below shows the output of running the script:

```
PS C:\Users\hariv\OneDrive\Documents\SRM\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRM/3 year II sem\AI_Assistant_coding\assignment_2.5.py"
Factorial of 5 is: 120
PS C:\Users\hariv\OneDrive\Documents\SRM\3 year II sem\AI_Assistant_coding> []
```

#### Prompt 3.2 :

Write factorial function in python using recursion

#### Code & Output:



The screenshot shows the Microsoft Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, Agents, Editor, and AI\_Assistant\_coding. The left sidebar shows a file tree with files like assignment\_1.3.py, assignment\_1.5.py, assignment\_2.5.py, and assignment\_2.5.py (the active tab). The main editor area contains the following Python code:

```
#Task 3
#prompt: 3.2 - Write factorial function in Python using recursion.
def recursive_factorial(n):
    """Calculate the factorial of a non-negative integer n using recursion."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1:
        return 1
    else:
        return n * recursive_factorial(n - 1)
print("Recursive factorial of 5 is:", recursive_factorial(5))
```

The terminal below shows the output of running the script:

```
PS C:\Users\hariv\OneDrive\Documents\SRM\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRM/3 year II sem\AI_Assistant_coding\assignment_2.5.py"
Recursive factorial of 5 is: 120
PS C:\Users\hariv\OneDrive\Documents\SRM\3 year II sem\AI_Assistant_coding> []
```

### Prompt 3.3:

Write an optional factorial function with comments.

### Code & Output:

The screenshot shows the AI Assistant Coding interface. The left sidebar lists files: AI\_ASSISTANT\_CODING, assignment\_1.3, assignment\_1.3.py, assignment\_1.3.pptx, assignment\_1.3.pdf, and Lab\_Assignment\_1.3.docx. The main area displays the following Python code:

```
#Task 3
#prompt 3.3 - Write an optimized Factorial function with comments.

def optimized_factorial(n, _cache={}):
    """
    Calculate the factorial of a non-negative integer n efficiently using memoization.

    Args:
        n (int): The non-negative integer for which to compute the factorial.
        _cache (dict): Internal cache to store previously computed factorials.

    Returns:
        int: The factorial of n.

    Raises:
        ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    # Base case: 0! and 1! are 1.
    if n == 0 or n == 1:
        _cache[n] = 1
        return 1
    # Return cached value if already computed.
    if n in _cache:
        return _cache[n]
    # Compute, cache and return the result.
    _cache[n] = n * optimized_factorial(n - 1, _cache)
    return _cache[n]

print("Optimized factorial of 5 is:", optimized_factorial(5))
```

The terminal below shows the output of running the script:

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python n3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_2.5.py"
Optimized factorial of 5 is: 120
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> []
```

### Prompt 3.4:

Write a factorial function in a single line.

### Code & Output:

The screenshot shows the AI Assistant Coding interface. The left sidebar lists files: AI\_ASSISTANT\_CODING, assignment\_1.3, assignment\_1.3.py, assignment\_1.3.pptx, assignment\_1.3.pdf, assignment\_2.5.py, Assignment\_1.5, Assignment\_1.5.pdf, and Lab\_Assignment\_1.3.docx. The main area displays the following Python code:

```
#prompt 3.4 - Write a factorial function in a single line.

factorial_single_line = lambda n: 1 if n == 0 else n * factorial_single_line(n - 1)

print("Single-line factorial of 5 is:", factorial_single_line(5))
```

The terminal below shows the output of running the script:

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python n3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_2.5.py"
Single-line factorial of 5 is: 120
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> []
```

### Prompt 3.5:

Write production grade factorial function with typing and docstring.

### Code & Output:

The screenshot shows the AI Assistant Coding application. The left sidebar lists files in the workspace, including 'AI\_Assistant\_CODING', 'Assignment-1\_5\_2303A51266.docx', 'AI\_Assistant\_coding.code-workspace', 'assignment\_1.5.py', 'assignment\_1.5.py', 'assignment\_2.5.py', and 'Assignment-1\_5\_2303A51266.pdf'. The main area displays a Python script named 'assignment\_2.5.py' with the following code:

```

#Task 3
#prompt: 3.5 - Write production grade factorial function with typing and docstring.
def factorial(n: int) -> int:
    """
    Calculate the Factorial of a non-negative integer n.
    Args:
        n (int): The non-negative integer whose factorial is to be computed.
    Returns:
        int: The factorial of n.
    Raises:
        ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
print("Production grade factorial of 5 is:", factorial(5))

```

The terminal below shows the command being run and its output:

```

PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents\SRU\3 year II sem\AI_Assistant_coding\assignment_2.5.py"
● Production grade factorial of 5 is: 120
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>

```

### Explanation:

In this task, I asked Cursor to generate the factorial program using different prompts. Even though the problem was the same, the tool produced different versions of the code depending on how I asked. First, it gave a simple loop-based factorial, then a recursive version when I mentioned recursion, and later an optimized one when I asked for better performance. It also handled a single-line lambda version and finally a production-style version with typing and docstrings. This shows that the tool understands prompt details and adjusts the code style, structure, and features based on how clearly the prompt is written.

### Task 4:

Compare Gemini, Copilot, and Cursor AI for usability and code quality.

#### Short reflection:

Gemini, Copilot, and Cursor AI each help with coding in different ways, especially when it comes to usability and code quality. Gemini is very good at explaining ideas and showing clear, readable examples. It helps you understand how code works, but it usually sticks to simple and safe patterns and does not easily adjust to different styles or advanced needs. This makes it better for learning than for serious development work.

GitHub Copilot is designed to make coding faster inside an editor. It completes lines and suggests blocks of code based on what you start typing. This improves speed, but it does not really think about whether the logic is correct or well designed. If you make a mistake, Copilot will continue building on that mistake.

Cursor AI is the most flexible and powerful of the three. It changes the way it writes code based on how the problem is asked. The same task can result in different styles such as simple,

optimized, or production-level code. This shows that Cursor understands the intent behind the prompt and produces better structured and more professional code.

Because of this, Cursor AI is the best overall choice. It combines good usability with high-quality, adaptable code, making it the most suitable tool for real-world software development.