

AI Assisted Coding

Assignment 2.5

Name: V.Harivamsh

Hall ticket no: 2303A51266

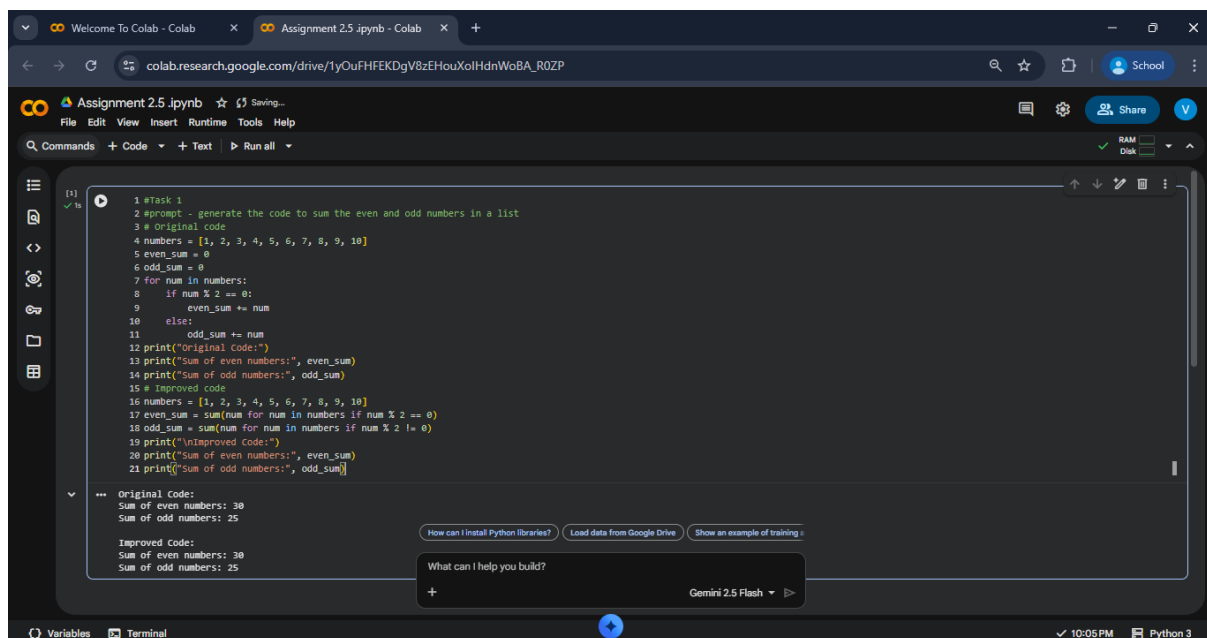
Batch no: 19

Task 1:

Prompt:

Generate the code to sum of even and odd numbers in a list

Code& Output:



```
1 #Task 1
2 #prompt - generate the code to sum the even and odd numbers in a list
3 # original code
4 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
5 even_sum = 0
6 odd_sum = 0
7 for num in numbers:
8     if num % 2 == 0:
9         even_sum += num
10    else:
11        odd_sum += num
12 print("Original Code:")
13 print("Sum of even numbers:", even_sum)
14 print("Sum of odd numbers:", odd_sum)
15 # Improved code
16 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
17 even_sum = sum(num for num in numbers if num % 2 == 0)
18 odd_sum = sum(num for num in numbers if num % 2 != 0)
19 print("\nImproved Code:")
20 print("Sum of even numbers:", even_sum)
21 print("Sum of odd numbers:", odd_sum)
```

Original Code:
Sum of even numbers: 30
Sum of odd numbers: 25

Improved Code:
Sum of even numbers: 30
Sum of odd numbers: 25

Explanation:

The original code uses a standard for-loop combined with if/else checks to go through the list and add up values. It works correctly but ends up being longer and more manual.

The updated version takes advantage of Python's built-in features:

- **Generator Expression:** Instead of building temporary lists, it uses a generator (number for number in my_list if condition) directly inside `sum()`, which saves memory because values are produced as needed.
- **sum() Function:** Python's `sum()` is a fast, built-in tool designed specifically for adding up elements from an iterable.

Overall, this rewritten method is cleaner, shorter, and more in line with typical Python style. It's also easier to read for experienced Python users, especially when dealing with simple calculations like this.

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 # We need this module to use the value of Pi (3.14159...)

def calculate_area(shape_type, dimension1, dimension2=0):
    """
    Function to calculate the area of a Circle, Rectangle, or Triangle.

    Parameters:
    - shape_type (str): The name of the shape (e.g., 'circle', 'rectangle',
    'triangle').
    - dimension1 (float): The first dimension (Radius for Circle, Length for
    Rectangle, Base for Triangle).
    - dimension2 (float): The second dimension (Width for Rectangle, Height for
    Triangle). Default is 0 for Circle.
    """

    area = 0 # Initialize variable to store the result

    # 1. Logic for Circle
    # Formula: Area = Pi * r^2
    if shape_type == "circle":
        radius = dimension1
        area = math.pi * (radius ** 2)
```

```

        print(f"Calculated area for Circle with radius {radius}")

# 2. Logic for Rectangle
# Formula: Area = Length * Width
elif shape_type == "rectangle":
    length = dimension1
    width = dimension2
    area = length * width

    print(f"Calculated area for Rectangle with length {length} and width
{width}")

# 3. Logic for Triangle
# Formula: Area = 0.5 * Base * Height
elif shape_type == "triangle":
    base = dimension1
    height = dimension2
    area = 0.5 * base * height

    print(f"Calculated area for Triangle with base {base} and height
{height}")

# 4. Error handling for unknown shapes
else:
    print(f"Error: The shape '{shape_type}' is not supported.")
    return None

return area

# --- Driver Code (Testing the function) ---
print("--- Area Calculator Results ---")

```

```

# Test Case 1: Circle

# We only pass one dimension (radius) because dimension2 defaults to 0
circle_area = calculate_area("circle", 5)
print(f"Result: {circle_area:.2f}\n") # .2f restricts output to 2 decimal places


# Test Case 2: Rectangle

# We pass length (10) and width (5)
rect_area = calculate_area("rectangle", 10, 5)
print(f"Result: {rect_area:.2f}\n")


# Test Case 3: Triangle

# We pass base (10) and height (5)
tri_area = calculate_area("triangle", 10, 5)
print(f"Result: {tri_area:.2f}\n")

```

Output:

```

--- Area Calculator Results ---

Calculated area for Circle with radius 5
Result: 78.54

Calculated area for Rectangle with length 10 and width 5
Result: 50.00

Calculated area for Triangle with base 10 and height 5
Result: 25.00

```

Explanation:

1. The `calculate_area` function is used to find the area of circles, rectangles, and triangles using just one function instead of writing three separate ones.
2. The `math` module is imported so we can use `math.pi`, which gives us the value of Pi for circle area calculations.
3. Depending on the shape you give, the function applies the right formula — circles use $\pi \times \text{radius}^2$, rectangles use $\text{length} \times \text{width}$, and triangles use $0.5 \times \text{base} \times \text{height}$.

4. If the user enters a shape that the function doesn't recognize, it shows an error message instead of crashing.
5. The sample test cases at the end show how the function works, and the results are printed with two decimal places to look clean.

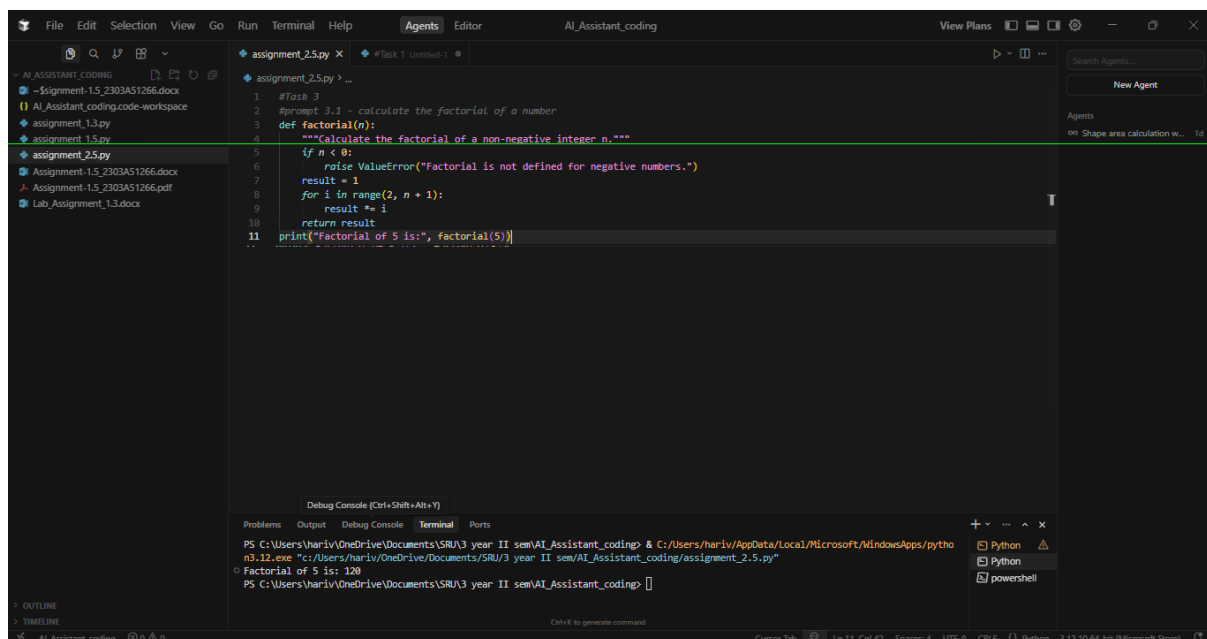
Task 3:

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 a code editor with a file explorer on the left, a code editor in the center, and a debug console at the bottom. The code editor contains a Python file named `assignment_2.5.py` with the following code:

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

The debug console at the bottom shows the output of the code:

```
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"
Factorial of 5 is: 120
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

Prompt 3.2 :

Write factorial function in python using recursion

Code & Output:

The screenshot shows the Visual Studio Code editor with a file named `assignment_2.5.py`. The code is a Python script that implements a recursive factorial function. The prompt in the code is: `#prompt 3.2 - Write factorial function in Python using recursion.` The function `recursive_factorial(n)` is defined with a docstring: `'''Calculate the factorial of a non-negative integer n using recursion.'''`. The function uses a `raise ValueError` for negative numbers and a base case for `n == 0` or `n == 1`. The output in the terminal shows the result of `recursive_factorial(5)` as 120.

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

Terminal 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"
Recursive factorial of 5 is: 120
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> |
```

Prompt 3.3:

Write an optional factorial function with comments.

Code & Output:

The screenshot shows the Visual Studio Code editor with a file named `assignment_2.5.py`. The code is a Python script that implements an optimized factorial function using memoization. The prompt in the code is: `#prompt 3.3 - Write an optimized factorial function with comments.` The function `optimized_factorial(n, _cache=())` is defined with a docstring: `'''Calculate the factorial of a non-negative integer n efficiently using memoization.'''`. The function includes a docstring for the `_cache` parameter and a docstring for the return value. The function uses a `raise ValueError` for negative numbers and a base case for `n == 0` or `n == 1`. The function also uses a `_cache` dictionary to store previously computed factorials. The output in the terminal shows the result of `optimized_factorial(5)` as 120.

```
1 #Task 3
2 #prompt 3.3 - Write an optimized factorial function with comments.
3 def optimized_factorial(n, _cache=()):
4     """Calculate the factorial of a non-negative integer n efficiently using memoization.
5     Args:
6         n (int): The non-negative integer for which to compute the factorial.
7         _cache (dict): Internal cache to store previously computed factorials.
8     Returns:
9         int: The factorial of n.
10    Raises:
11        ValueError: If n is negative.
12    """
13    if n < 0:
14        raise ValueError("Factorial is not defined for negative numbers.")
15    # Base case: 0! and 1! are 1.
16    if n == 0 or n == 1:
17        _cache[n] = 1
18        return 1
19    # Return cached value if already computed.
20    if n in _cache:
21        return _cache[n]
22    # Compute, cache and return the result.
23    _cache[n] = n * optimized_factorial(n - 1, _cache)
24    return _cache[n]
25 print("Optimized factorial of 5 is:", optimized_factorial(5))
```

Terminal 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"
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 a code editor with a file explorer on the left containing files like 'AI_Assistant_coding', 'assignment_1.5_2303A51266.docx', and 'assignment_2.5.py'. The main editor area displays a Python script for 'assignment_2.5.py' with the following code:

```
1 #prompt 3.4 - Write a factorial function in a single line.  
2 factorial_single_line = lambda n: 1 if n == 0 else n * factorial_single_line(n - 1)  
3 print("Single-line factorial of 5 is:", factorial_single_line(5))
```

The bottom panel shows the terminal 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"  
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 same code editor with a more complex Python script for 'assignment_2.5.py' that includes typing and docstrings:

```
1 #Task 3  
2 #prompt 3.5 - Write production grade factorial function with typing and docstring.  
3 def factorial(n: int) -> int:  
4     """  
5     Calculate the factorial of a non-negative integer n.  
6     Args:  
7         n (int): The non-negative integer whose factorial is to be computed.  
8     Returns:  
9         int: The factorial of n.  
10    Raises:  
11        ValueError: If n is negative.  
12    """  
13    if n < 0:  
14        raise ValueError("Factorial is not defined for negative numbers.")  
15    result = 1  
16    for i in range(2, n + 1):  
17        result *= i  
18    return result  
19  
20 print("Production grade factorial of 5 is:", factorial(5))
```

The terminal output is identical to the previous screenshot, showing the successful execution of the script and the output 'Production grade factorial of 5 is: 120'.

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 all support programmers, but they each focus on different strengths. Gemini works well as a learning assistant because it explains concepts clearly and provides clean, understandable examples. However, it tends to stay within basic, beginner-friendly patterns and doesn't easily adapt to more advanced coding styles or specialized requirements, so it fits educational use more than serious development work.

GitHub Copilot focuses on speed inside the code editor. It predicts and completes lines or entire blocks of code as you type, which makes development faster. The downside is that it doesn't check whether the overall logic is correct—if the user's approach is flawed, Copilot typically continues following that mistake without fixing it.

Cursor AI stands out because it can adjust the way it writes code depending on how the problem is described. The same prompt can produce beginner-style code, optimized code, or production-ready code, showing that Cursor understands both context and intention. This leads to cleaner structure and more practical output for real projects.

For these reasons, Cursor AI offers the most well-rounded experience. It combines ease of use with strong code quality and adaptability, making it the most suitable tool for real-world software development.