AI Assisted Coding

Assignment-1

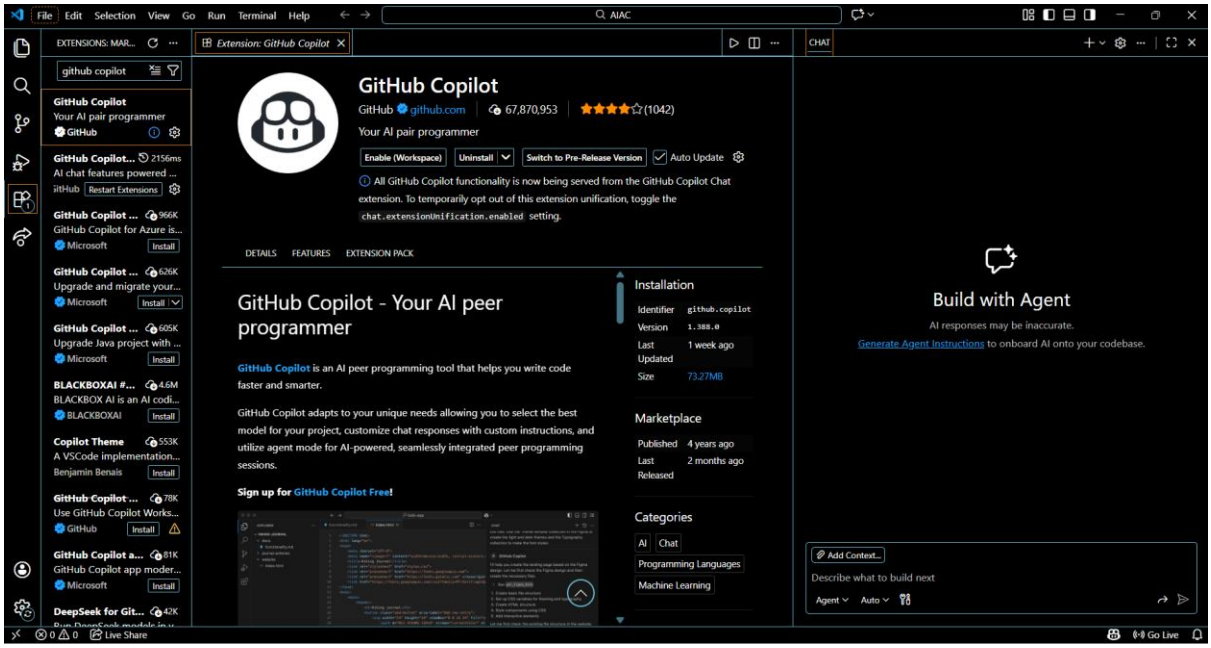
Name: Harshitha Guda

H.T.No:2303A51102

Lab 1: Environment Setup – GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow

Task 0

● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



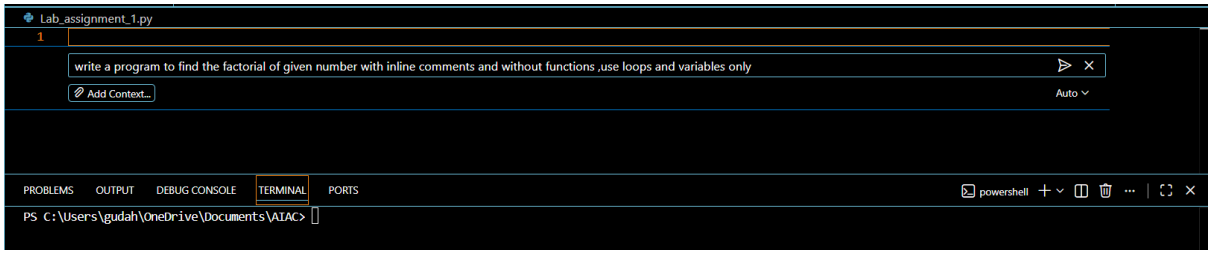
Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

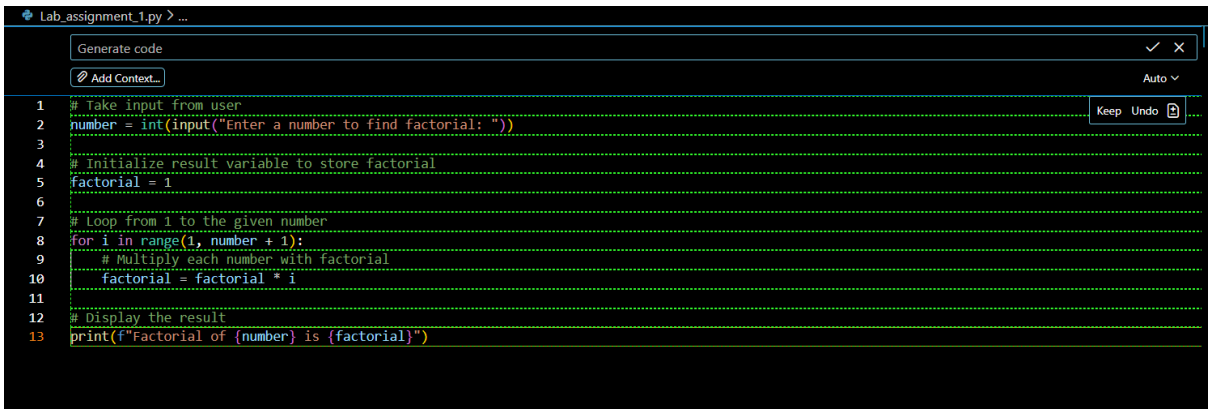
• Scenario

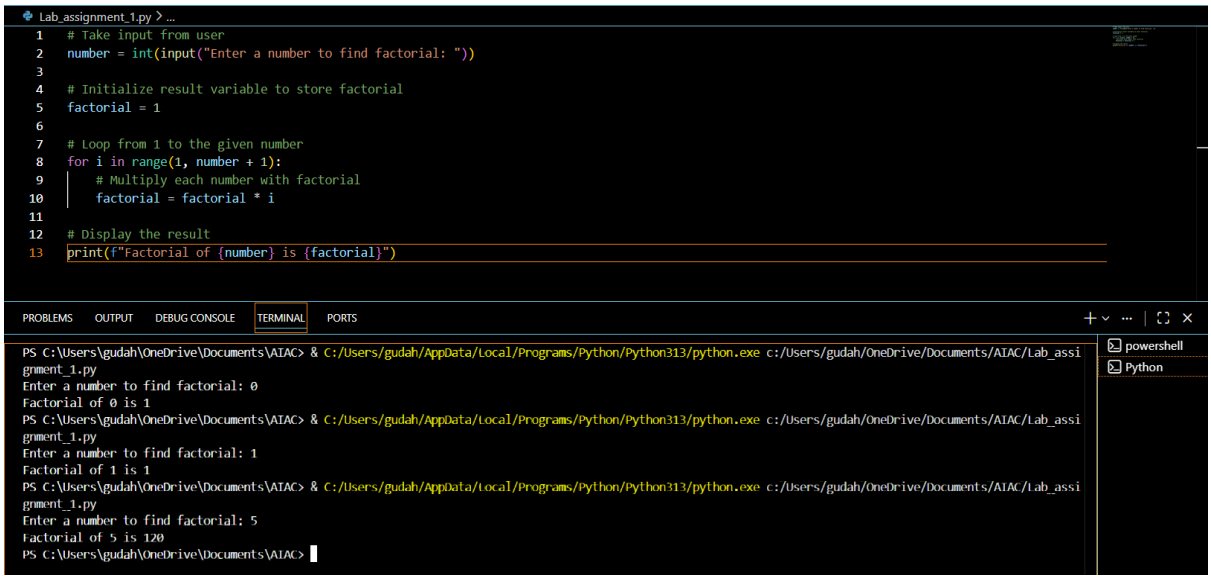
You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

• Task Description

Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.







Brief reflection :

➢ How helpful was Copilot for a beginner?

Copilot is very helpful for a beginner because it gave instant code suggestions and reduced syntax errors. It made writing Python programs easier and faster, and helped understand the basic structure of code.

➢ Did it follow best practices automatically?

Yes, Copilot usually followed good coding practices like proper indentation and clear variable names. However, the generated code still needed to be reviewed and understood by the user.

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

Scenario

Your team lead asks you to review AI-generated code before committing it to a shared repository.

Task Description

Analyse the code generated in Task 1 and use Copilot again to:

➢ Reduce unnecessary variables

➢ Improve loop clarity

➢ Enhance readability and efficiency

```python
#Optimize the above code by reducing unnecessary variables,improve loop clarity and add error handcling for negative inputs
# and non-integer values. and enhance readability and effeciency
try:
    number = int(input("Enter a non-negative integer to find its factorial: "))
    if number < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        factorial = 1
        for i in range(2, number + 1):  # Start from 2 as multiplying by 1 is redundant
            factorial *= i
        print(f"Factorial of {number} is {factorial}")
except ValueError:
    print("Invalid input! Please enter a non-negative integer.")

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: 9
Factorial of 9 is 362880
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: -4
Factorial is not defined for negative numbers.
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: d
Invalid input! Please enter a non-negative integer.
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

Side-by-Side Comparision:

First Code (Basic Version)

- Takes a number from the user and assumes the input is valid.
- Uses a loop from **1 to the number** to calculate factorial.
- Multiplies every value, including multiplying by 1 (which is unnecessary).
- Does **not handle negative numbers** or wrong inputs (like letters).
- Simple and easy to understand, but not very safe or efficient.

Second Code (Optimized Version)

- Uses **try–except** to handle wrong inputs like letters or symbols.
- Checks if the number is **negative** and shows a proper message.
- Starts the loop from **2**, avoiding unnecessary multiplication by 1.

- Uses clearer messages and cleaner logic.
- More **efficient, readable, and user-friendly**.

What was improved?
The code was improved by adding error handling for invalid and negative inputs. Unnecessary multiplication by 1 was removed, and the loop was made clearer. User messages were also improved for better understanding.

Why the new version is better?
The new version is more readable because the logic is clean and easy to follow. It performs better by avoiding unnecessary operations. It is also more maintainable since errors are handled properly and the code is safer for real-world use.

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

Scenario

The same logic now needs to be reused in multiple scripts.

Task Description

Use GitHub Copilot to generate a modular version of the program by:

➢ Creating a user-defined function

➢ Calling the function from the main block

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs

Without Functions)

❖ Scenario

As part of a code review meeting, you are asked to justify design choices.

❖ Task Description

Compare the non-function and function-based Copilot-generated

programs on the following criteria:

➢ Logic clarity

➢ Reusability

➢ Debugging ease

➢ Suitability for large projects

➢ AI dependency risk

Logic Clarity:
In the non-function factorial code, all steps—input, loop, calculation, and output—are written together. This is easy to understand for a simple factorial program, but the logic becomes unclear if more features are added. In the function-based factorial code, the factorial calculation is placed inside a function. This clearly separates "what the program does" from "how it does it," making the logic easier to read.

Reusability:
Procedural factorial code cannot be reused easily. If factorial is needed again, the same loop must be rewritten. In the modular version, the factorial function can be reused anywhere by calling it. This makes the function-based approach more flexible and efficient.

Debugging Ease:
Debugging is harder in non-function code because all logic is in one block. Finding mistakes takes more time. In the function-based factorial program, errors can be traced directly to the factorial function, making debugging simpler and faster.

Suitability for Large Projects:
Procedural factorial code is suitable only for small examples or practice programs. In large projects, it becomes difficult to manage. The function-based factorial code fits better in large projects because it is organized, easy to maintain, and supports teamwork.

AI Dependency Risk:
When Copilot generates procedural code, users may copy it without understanding. Function-based code encourages better structure and logical thinking, reducing over-dependence on AI tools.

Conclusion:
For factorial programs, procedural code is simple but limited. Function-based code offers better clarity, reuse, debugging, and long-term usefulness, making it the better design choice.

Task 5: AI-Generated Iterative vs Recursive Thinking

❖ Scenario

Your mentor wants to test how well AI understands different

computational paradigms.

❖ Task Description

Prompt Copilot to generate:

An iterative version of the logic

A recursive version of the same logic

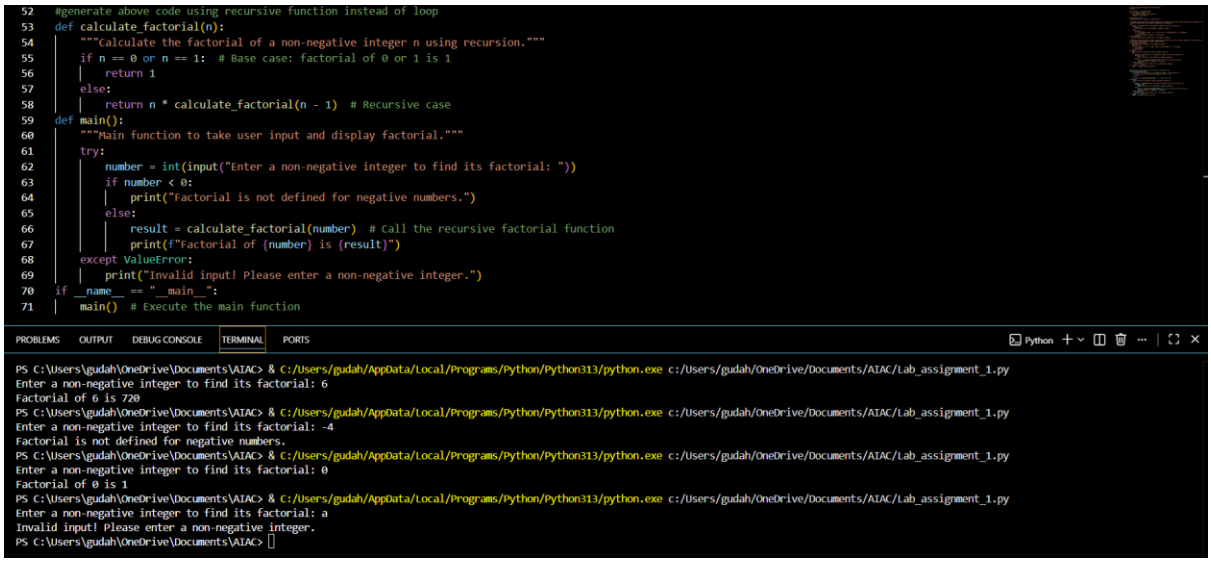
Iterative version

```python
#generate iterative version of the above code
def calculate_factorial(n):
    """Calculate the factorial of a non-negative integer n using iteration."""
    factorial = 1
    for i in range(2, n + 1):  # Start from 2 as multiplying by 1 is redundant
        factorial *= i
    return factorial
def main():
    """Main function to take user input and display factorial."""
    try:
        number = int(input("Enter a non-negative integer to find its factorial: "))
        if number < 0:
            print("Factorial is not defined for negative numbers.")
        else:
            result = calculate_factorial(number)  # Call the iterative factorial function
            print(f"Factorial of {number} is {result}")
    except ValueError:
        print("Invalid input! Please enter a non-negative integer.")
if __name__ == "__main__":
    main()  # Execute the main function
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: 6
Factorial of 6 is 720
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: -8
Factorial is not defined for negative numbers.
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: 0
Factorial of 0 is 1
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: a
Invalid input! Please enter a non-negative integer.
```

In the **iterative version**, the program takes input from the user, checks if the number is valid, and then uses a loop to multiply numbers from 2 up to the given number. The result is stored in a variable and printed.

Recursive version



```python
#generate above code using recursive function instead of loop
def calculate_factorial(n):
    """Calculate the factorial of a non-negative integer n using recursion."""
    if n == 0 or n == 1:  # Base case: factorial of 0 or 1 is 1
        return 1
    else:
        return n * calculate_factorial(n - 1)  # Recursive case
def main():
    """Main function to take user input and display factorial."""
    try:
        number = int(input("Enter a non-negative integer to find its factorial: "))
        if number < 0:
            print("Factorial is not defined for negative numbers.")
        else:
            result = calculate_factorial(number)  # Call the recursive factorial function
            print(f"Factorial of {number} is {result}")
    except ValueError:
        print("Invalid input! Please enter a non-negative integer.")
if __name__ == "__main__":
    main()  # Execute the main function
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: 6
Factorial of 6 is 720
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: -4
Factorial is not defined for negative numbers.
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: 0
Factorial of 0 is 1
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Users/gudah/AppData/Local/Programs/Python/Python313/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_1.py
Enter a non-negative integer to find its factorial: a
Invalid input! Please enter a non-negative integer.
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

In the **recursive version**, the program again takes input and validates it. The factorial function calls itself repeatedly, reducing the number by 1 each time, until it reaches the base case (0 or 1). Then it returns the result step by step back to the main function.

Comparison

1. Readability

- Iterative code is easier to understand for beginners because it uses a simple loop.
- Recursive code is shorter and mathematically clear, but harder to understand at first.

2. Stack Usage

- Iterative version uses very little memory because it does not store function calls.

- Recursive version uses more memory because each function call is stored on the call stack.

3. Performance Implications

- Iterative factorial is faster and more efficient for large numbers.
- Recursive factorial is slower due to repeated function calls.

4. When Recursion Is Not Recommended

- Recursion is not recommended for very large inputs.
- It can cause stack overflow errors.
- Iteration is better when performance and memory efficiency are important.