

# AI Assisted Coding

N. Goutham

batch 04

2303A51209

07-01-2026

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

### Lab Objectives:

- To install and configure GitHub Copilot in Visual Studio Code.
- To explore AI-assisted code generation using GitHub Copilot.
- To analyze the accuracy and effectiveness of Copilot's code suggestions.
- To understand prompt-based programming using comments and code context

### Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Set up GitHub Copilot in VS Code successfully.
- Use inline comments and context to generate code with Copilot.
- Evaluate AI-generated code for correctness and readability.
- Compare code suggestions based on different prompts and programming styles.

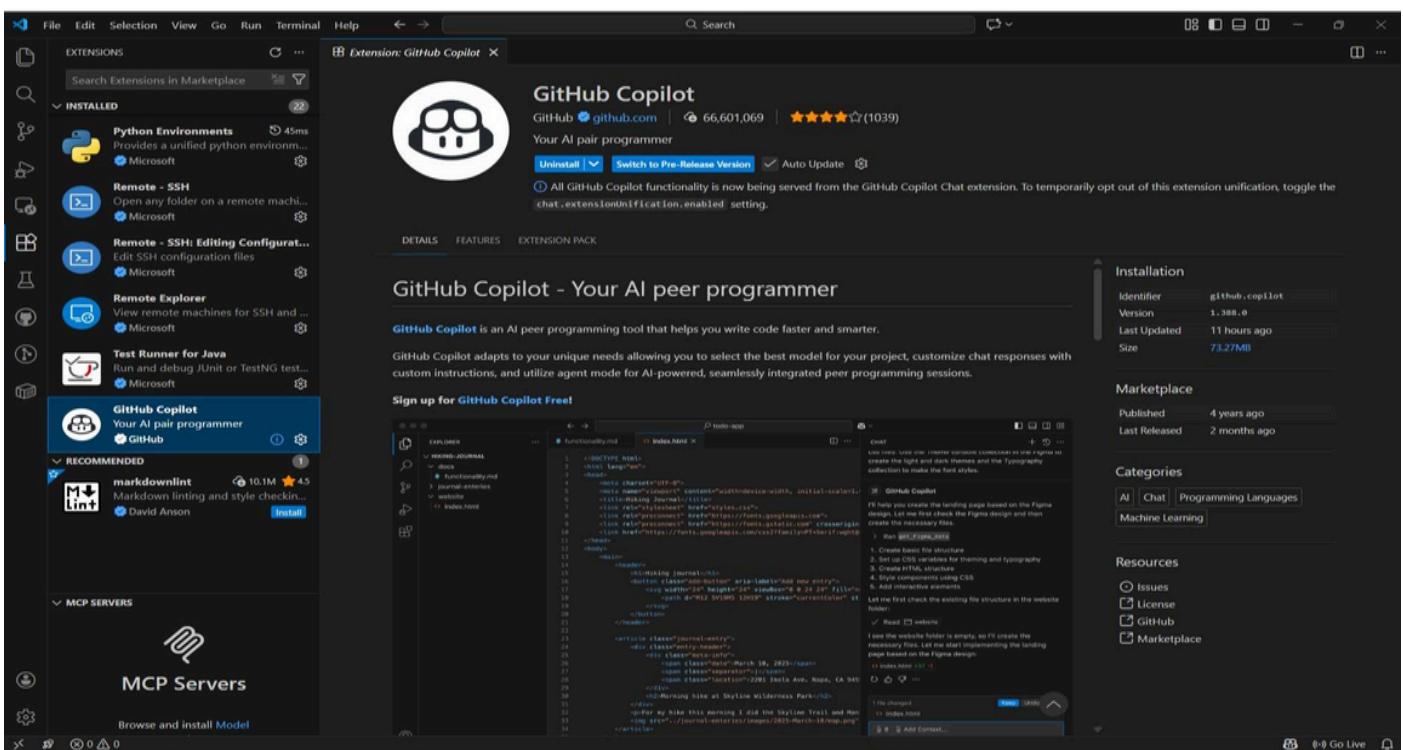
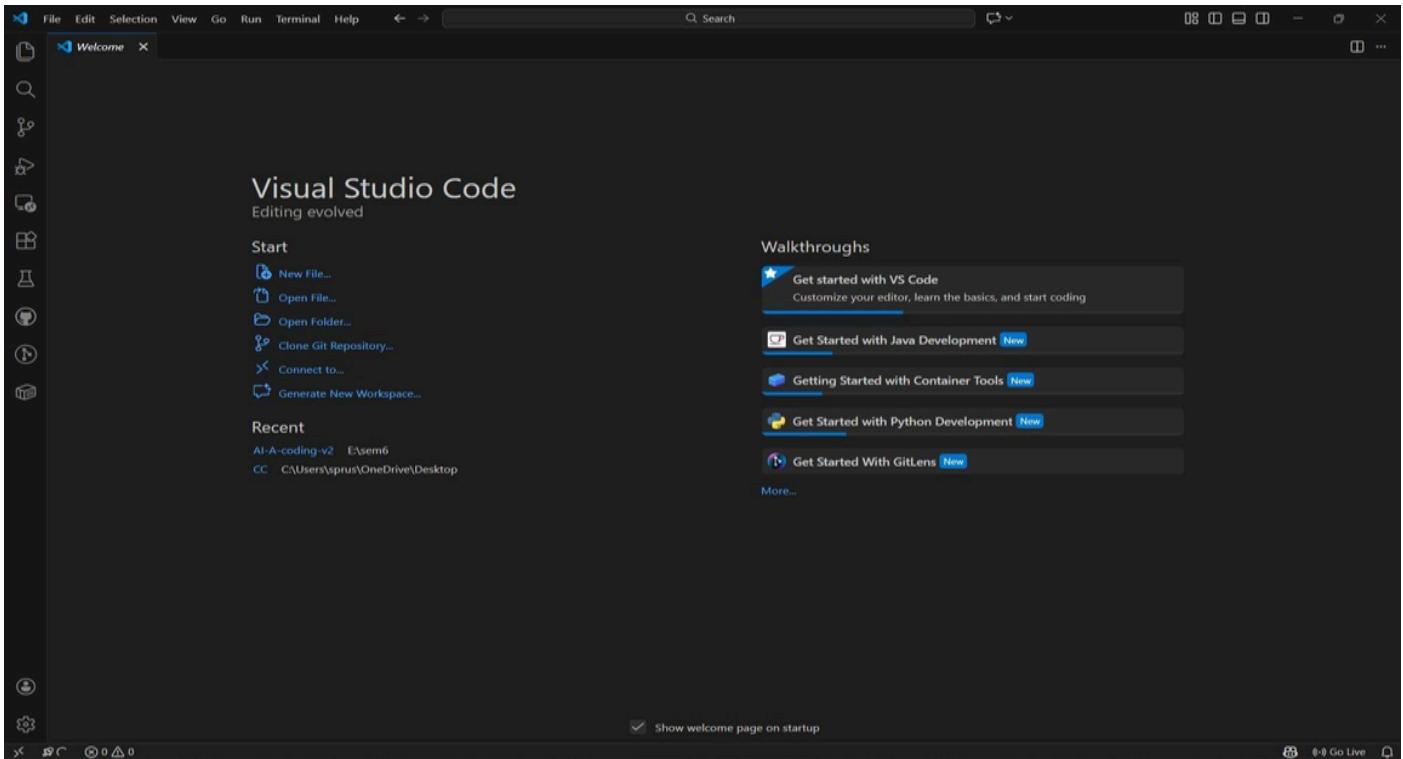
### Task 0

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

### Expected Output

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

## Task 0: Environment Setup:-



## 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.

- Constraint:

- Do not define any custom function
- Logic must be implemented using loops and variables only

- Expected Deliverables

- A working Python program generated with Copilot assistance

- Screenshot(s) showing:

- The prompt you typed

- Copilot's suggestions

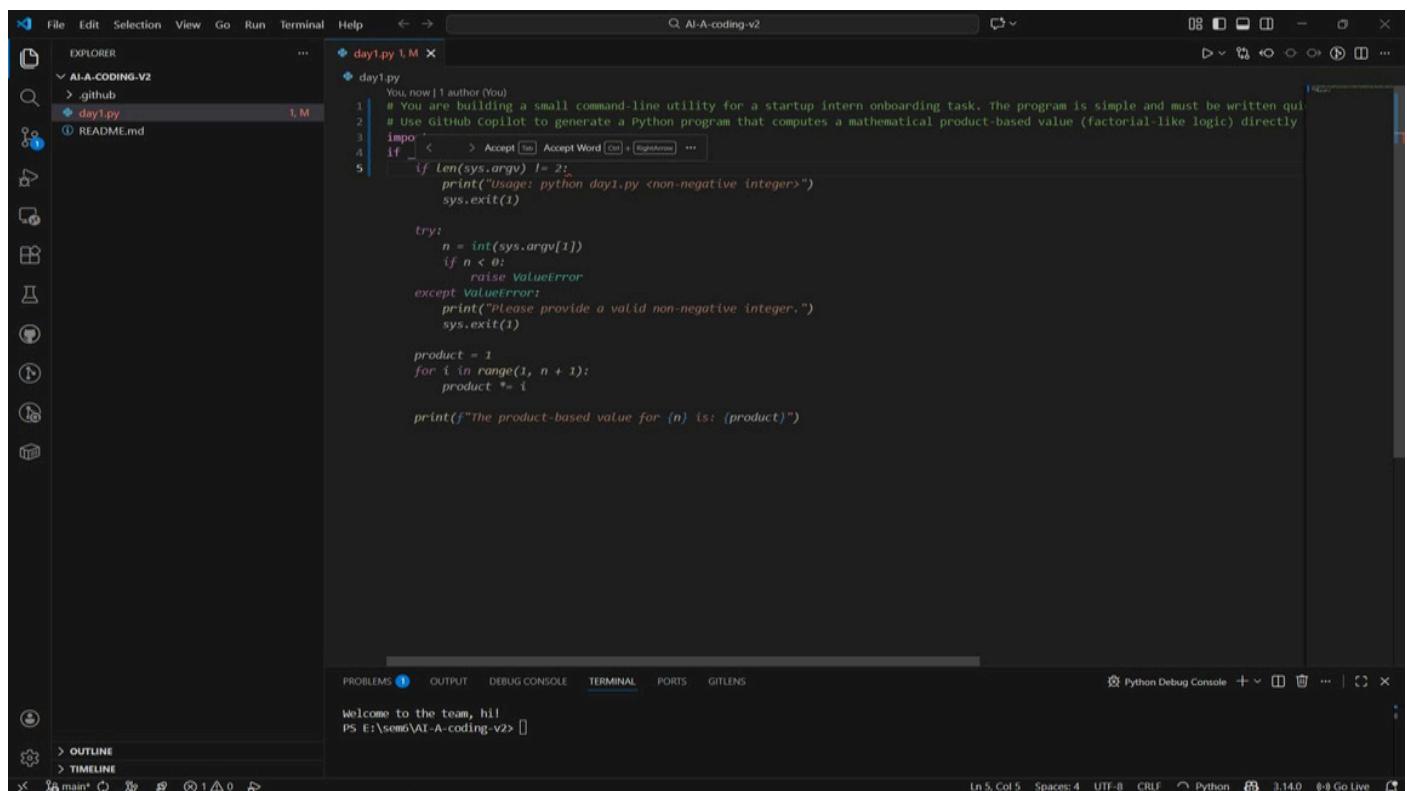
- Sample input/output screenshots

- Brief reflection (5-6 lines):

- How helpful was Copilot for a beginner?

- Did it follow best practices automatically?

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



The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows a folder named "AI-A-CODING-V2" containing ".github", "day1.py", and "README.md".
- Code Editor:** Displays the content of "day1.py". The code is as follows:

```
You, now | 1 author (You)
1 # You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly.
2 # 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.
3 import sys
4 if len(sys.argv) < 2:
5     print("Usage: python day1.py <non-negative integer>")
6     sys.exit(1)
7
8 try:
9     n = int(sys.argv[1])
10    if n < 0:
11        raise ValueError
12    except ValueError:
13        print("Please provide a valid non-negative integer.")
14        sys.exit(1)
15
16    product = 1
17    for i in range(1, n + 1):
18        product *= i
19
20    print(f"The product-based value for {n} is: {product}")
```

- Terminal:** Shows the command "PS E:\sem6\AI-A-coding-v2>".
- Status Bar:** Shows "Ln 5, Col 5" and "Python 3.14.0 Go Live".

The screenshot shows the VS Code interface with the following details:

- Editor:** The main editor window displays a Python file named `day1.py`. The code is a simple program that calculates the product of all non-negative integers up to a given input. It includes basic error handling for negative inputs and non-integer arguments.
- Terminal:** Below the editor is the terminal pane, which shows the execution of the code. It starts with the command `python day1.py 5`, followed by the output "The product-based value for 5 is: 120".
- Status Bar:** The status bar at the bottom indicates the file is 3.14.0 and shows other standard status information.

## Task 2: AI Code Optimization C Cleanup (Improving Efficiency)

### ❖ Scenario

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

### ❖ Task Description

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

- Reduce unnecessary variables
- Improve loop clarity
- Enhance readability and efficiency

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

### ❖ Expected Deliverables

- Original AI-generated code
- Optimized version of the same code

## ➤ Side-by-side comparison

### ➤ Written explanation:

- What was improved?
- Why the new version is better (readability, performance, maintainability).

## Task 2: AI Code Optimization C Cleanup (Improving Efficiency)

```
File Edit Selection View Go Run Terminal Help < > Q: AI-A-coding-v2
day1.py M task2.py 1.0 X
task2.py
1 # In Task 2, the AI-generated code from Task 1 was reviewed and optimized.
2 # Unnecessary loop iterations were removed by starting the loop from 2 instead of 1.
3 # Output text was simplified to improve readability.
4 # The code structure was kept flat to follow the no-function constraint.
5 # Overall, the optimized version is easier to read, slightly more efficient, and more maintainable while producing the same result.
6 import sys
7 if __name__ == "__main__":
8     if len(sys.argv) != 2:
9         print("Usage: python task2.py <non-negative integer>")
10        sys.exit(1)
11
12     try:
13         n = int(sys.argv[1])
14         if n < 0:
15             raise ValueError
16     except ValueError:
17         print("Please provide a valid non-negative integer.")
18         sys.exit(1)
19
20     product = 1
21     for i in range(2, n + 1):
22         product *= i
23
24     print(product)
```

```
File Edit Selection View Go Run Terminal Help < > Q: AI-A-coding-v2
day1.py M task2.py 2.0 X
task2.py
1 # In Task 2, the AI-generated code from Task 1 was reviewed and optimized.
2 # Unnecessary loop iterations were removed by starting the loop from 2 instead of 1.
3 # Output text was simplified to improve readability.
4 # The code structure was kept flat to follow the no-function constraint.
5 # Overall, the optimized version is easier to read, slightly more efficient, and more maintainable while producing the same result.
6 import sys
7 if __name__ == "__main__":
8     if len(sys.argv) != 2:
9         print('Usage: python task2.py <non-negative integers>')
10        sys.exit(1)
11
12     try:
13         n = int(sys.argv[1])
14         if n < 0:
15             raise ValueError
16     except ValueError:
17         print("Please provide a valid non-negative integer.")
18         sys.exit(1)
19
20     product = 1
21     for i in range(2, n + 1):
22         product *= i
23
24     print(product)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS E:\sem6\AI-A-coding-v2> python task2.py 5
120
PS E:\sem6\AI-A-coding-v2> |
```

Ln 24, Col 19 Spaces: 4 UTF-8 CRLF {} Python 3.14.0 ⓘ Go live

## 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

## ❖ Constraints

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

- Use meaningful function and variable names
- Include inline comments (preferably suggested by Copilot)

## ❖ Expected Deliverables

- AI-assisted function-based program

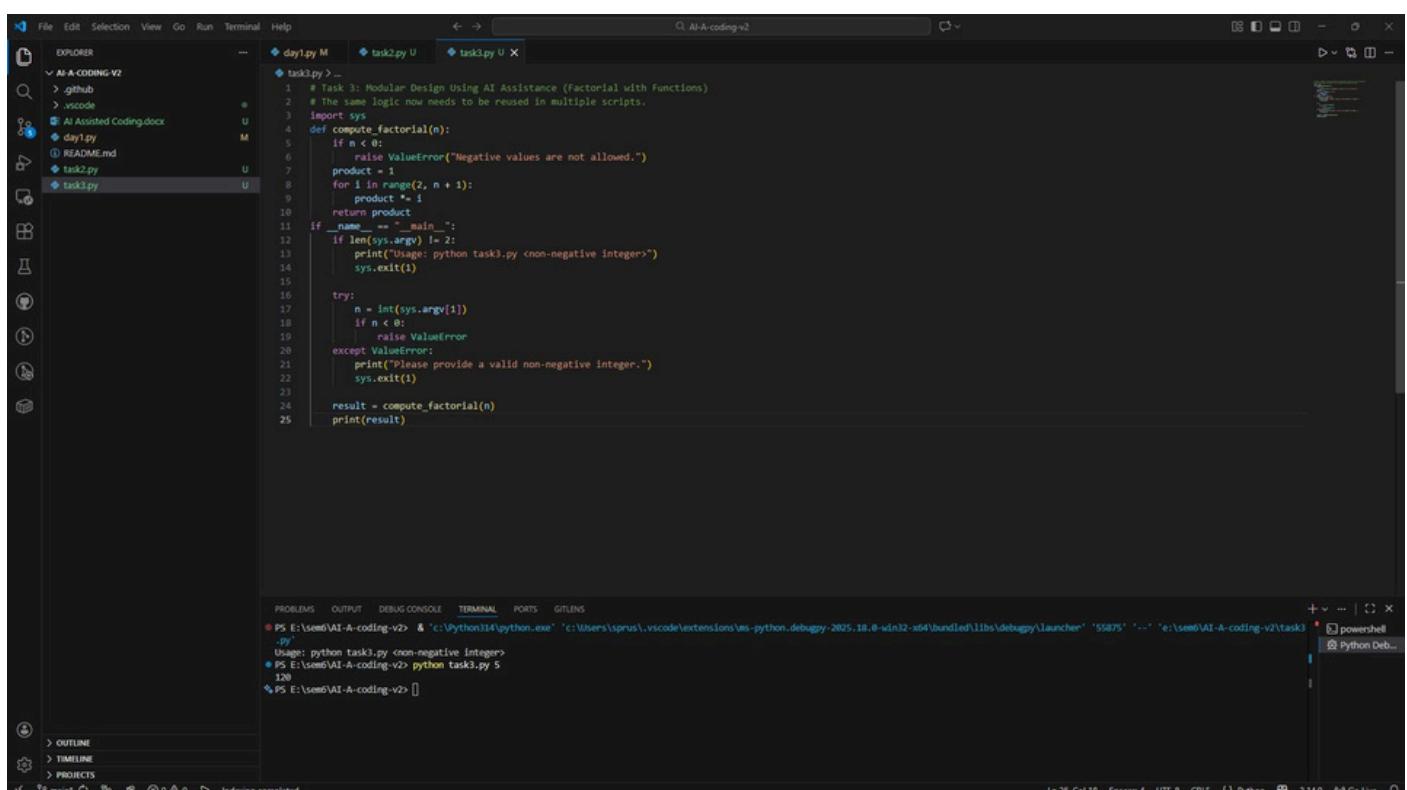
### ➤ Screenshots showing:

- Prompt evolution
- Copilot-generated function logic

### ➤ Sample inputs/outputs

### ➤ Short note:

### ○ How modularity improves reusability.



```
task3.py >-
1 # Task 3: Modular Design Using AI Assistance (Factorial with Functions).
2 # The same logic now needs to be reused in multiple scripts.
3 import sys
4 def compute_factorial(n):
5     if n < 0:
6         raise ValueError("Negative values are not allowed.")
7     product = 1
8     for i in range(2, n + 1):
9         product *= i
10    return product
11 if __name__ == "__main__":
12    if len(sys.argv) != 2:
13        print("Usage: python task3.py <non-negative integer>")
14        sys.exit(1)
15
16    try:
17        n = int(sys.argv[1])
18        if n < 0:
19            raise ValueError
20    except ValueError:
21        print("Please provide a valid non-negative integer.")
22        sys.exit(1)
23
24    result = compute_factorial(n)
25    print(result)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

PS E:\sem6\AI-A-coding-v2> & "c:\python314\python.exe" "c:\Users\spurush\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "55875" ... 'E:\sem6\AI-A-coding-v2\task3.py'
Usage: python task3.py <non-negative integer>
PS E:\sem6\AI-A-coding-v2> python task3.py
120
PS E:\sem6\AI-A-coding-v2> []

OUTLINE TIMELINE PROJECTS

Indexing completed.

**Short Note: How Modularity Improves Reusability** Modularity helps in reusability by helping separate logic in terms of different functions which may be reused in multiple programs. The factorial computation is put in a function which makes the code easier to maintain and test. If the logic has to be changed, changes can be made at one place without having any impact on the whole program. Modular code is also more readable and easier to work on in a team environment.

#### **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

##### **❖ Expected Deliverables**

Choose one:

- A comparison table
- OR
- A short technical report (300-400 words).

## Task 4: Comparative Analysis - Procedural vs Modular AI Code (With vs Without Functions)

Criteria	Procedural Code (Without Functions)	Modular Code (With Functions)
Logic Clarity	Logic is written in one continuous flow, which is easy to follow for very small programs but functions, making the purpose of each part easier to understand.	Logic is clearly separated into functions, making it easier to understand.
Reusability	logic is tightly coupled to the main execution across multiple scripts by importing the function.	Modular code can be reused across different projects by importing the module.
Debugging Ease	Debugging is difficult as all logic exists in one block, making it difficult to isolate issues. can be traced to specific functions.	Debugging is easier as logic is modularized and can be traced to specific functions.
Suitability for Large Projects	Well suited for large projects due to better modularity, readability, and maintainability.	Not suitable for large projects as it leads to lower maintainability and teamwork support.
AI Dependency Risk	Lower risk, as modular structure encourages understanding of individual components.	High risk, as beginners may copy AI-generated code without understanding the full flow.

## 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

### ❖ Constraints

Both implementations must produce identical outputs

Students must not manually write the code first

### ❖ Expected Deliverables

Two AI-generated implementations

Execution flow explanation (in your own words)

Comparison covering:

- Readability
- Stack usage
- Performance implications
- When recursion is not recommended.

## Task 5: AI-Generated Iterative vs Recursive Thinking

### Iterative Thinking -

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure under "AI-A CODING-V2" containing files: README.md, day1.py, task2.py, task3.py, and task5\_iterative.py.
- Code Editor:** Displays the content of task5\_iterative.py. The code is an iterative factorial calculator using a loop. It includes error handling for non-negative integers and command-line arguments.

```
task5_iterative.py
1 # iterative approach using a loop
2 # This method calculates factorial by repeatedly multiplying values
3
4 import sys
5
6 if __name__ == "__main__":
7     if len(sys.argv) != 2:
8         print("Usage: python task5_iterative.py <non-negative integer>")
9         sys.exit(1)
10
11     try:
12         n = int(sys.argv[1])
13         if n < 0:
14             raise ValueError
15     except ValueError:
16         print("Please provide a valid non-negative integer.")
17         sys.exit(1)
18
19     result = 1
20     for i in range(2, n + 1):
21         result *= i
22
23     print(result)
24
```

- Terminal:** Shows the command "python task5\_iterative.py 5" being run, resulting in the output "120".
- Status Bar:** Shows indexing completed and other system information like Ln 8, Col 36, Spaces 4, etc.

## Recursive thinking -

```
File Edit Selection View Go Run Terminal Help
AI-A-CODING-V2
task5_recursive.py >-
1 # Recursive approach using function calls
2 # The function calls itself until it reaches the base case
3
4 import sys
5
6 def factorial(n):
7     if n == 0 or n == 1:
8         return 1
9     return n * factorial(n - 1)
10
11 if __name__ == "__main__":
12     if len(sys.argv) != 2:
13         print("Usage: python task5_recursive.py <non-negative integer>")
14         sys.exit(1)
15
16 try:
17     n = int(sys.argv[1])
18     if n < 0:
19         raise ValueError
20 except ValueError:
21     print("Please provide a valid non-negative integer.")
22     sys.exit(1)
23
24 print(factorial(n))
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS E:\sem6\AI-A-CODING-V2> python task5_recursive.py 5
120
PS E:\sem6\AI-A-CODING-V2>
```

Ln 13, Col 45 Spaces: 4 UTF-8 CRLF Python 3.14.0 88 Go Live

### Aspect

### Readability

### Stack Usage

### Performance

### Error Risk

### When Not Recommended

### Iterative Approach

Easy to understand for beginners

Uses constant memory

Faster and memory efficient

Low

### Recursive Approach

More mathematical and elegant

Uses additional stack memory

Slower for large inputs

Risk of stack overflow

Not recommended for large input values

## Assignment -1.5

### AI Assisted Coding

nishath afroze

batch 04

2303A51212

09-01-2026

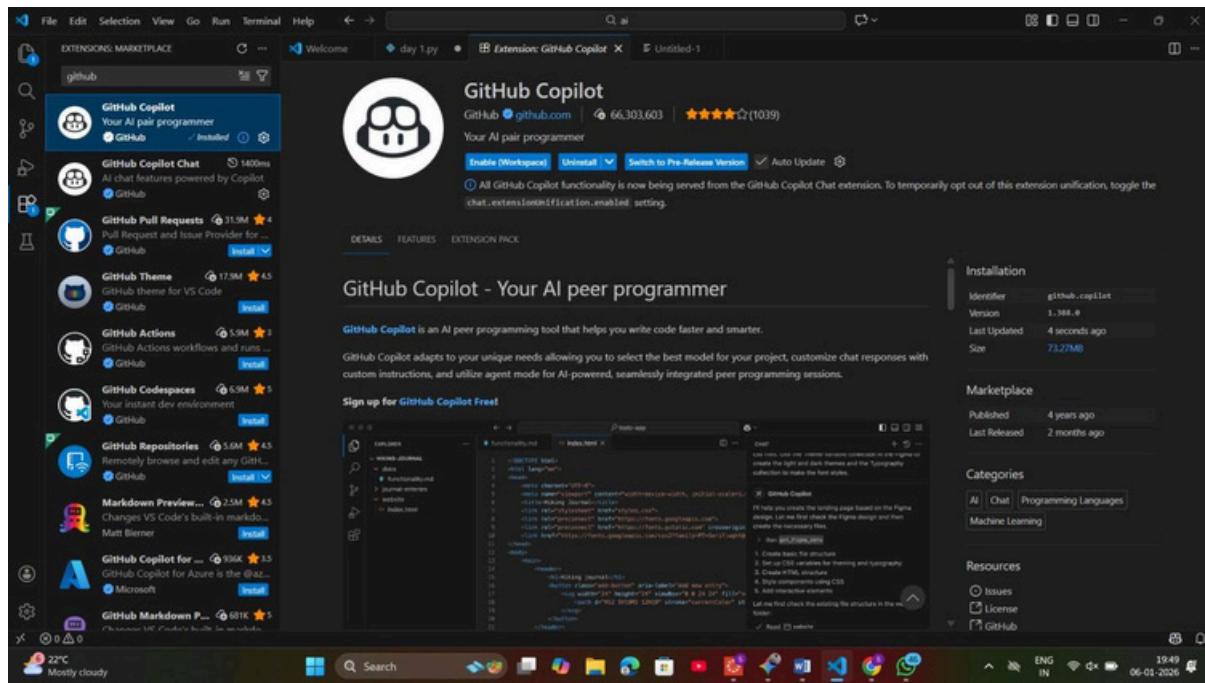
#### Task 0: Environment Setup:-

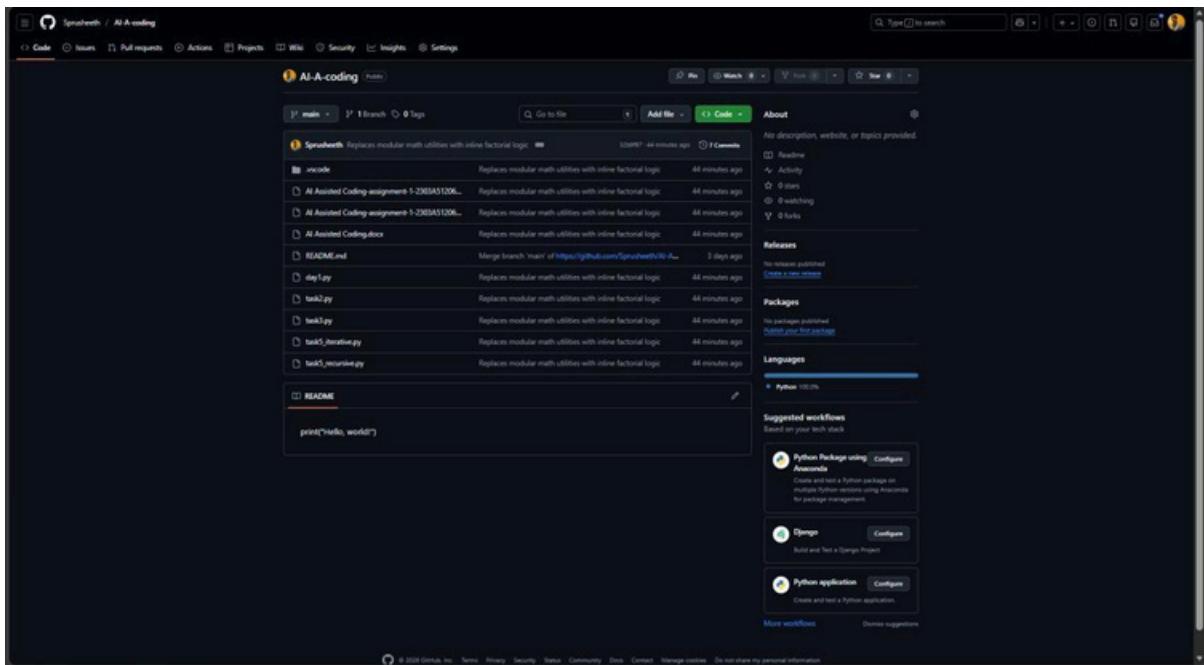
##### Task 0

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

##### Expected Output

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





## Task 1: Non-Modular Logic (Factorial):-

: AI-Generated Logic Without Modularization (String Reversal Without Functions)

### ❖ Scenario

You are developing a basic text-processing utility for a messaging application.

### ❖ Task Description

Use GitHub Copilot to generate a Python program that:

- Reverses a given string
- Accepts user input
- Implements the logic directly in the main code
- Does not use any user-defined functions

### ❖ Expected Output

- Correct reversed string
- Screenshots showing Copilot-generated code suggestions

## ➤ Sample inputs and outputs

The screenshot shows the VS Code interface with two windows. The top window displays the code file `task1.py` containing a script to reverse a string. The bottom window shows the terminal output where the script is run, accepting input and printing the original and reversed strings.

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
1 # Accepting user input
2 user_input = input("Enter a string to reverse: ")
3
4 # Initializing an empty string to store the result
5 reversed_string = ""
6
7 # logic to reverse the string using a loop
8 for i in range(len(user_input) - 1, -1, -1):
9     reversed_string += user_input[i]
10
11 # Printing the result
12 print("Original String:", user_input)
13 print("Reversed String:", reversed_string)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⚡ ... ✎
PS C:\Users\hp> & c:/users/hp/appdata/local/microsoft/windowsapps/python3.13.exe c:/users/hp/onedrive/desktop/ai/task1.py
Enter a string to reverse: 2 3 4 5 6
Original String: 2 3 4 5 6
Reversed String: 6 5 4 3 2
PS C:\Users\hp>
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\hp> & c:/users/hp/appdata/local/microsoft/windowsapps/python3.13.exe c:/users/hp/onedrive/desktop/ai/task1.py
Enter a string to reverse: 2 3 4 5 6
Original String: 2 3 4 5 6
Reversed String: 6 5 4 3 2
PS C:\Users\hp>
```

## Task 2: AI Code Optimization:- Efficiency & Logic Optimization

### (Readability Improvement)

#### ❖ Scenario

The code will be reviewed by other developers.

#### ❖ Task Description

Examine the Copilot-generated code from Task 1 and improve it by:

- Removing unnecessary variables
- Simplifying loop or indexing logic
- Improving readability
- Use Copilot prompts like:
  - “Simplify this string reversal code”
  - “Improve readability and efficiency”

**Hint:**

**Prompt Copilot with phrases like**

**“optimize this code”, “simplify logic”, or “make it more readable”**

❖ **Expected Output**

➤ **Original and optimized code versions**

➤ **Explanation of how the improvements reduce time complexity**

The screenshot shows the Visual Studio Code interface. The left pane displays a file named 'task1.py' with the following code:

```
C:\Users\hp>OneDrive\Desktop>ai>task1.py>_
1 user_input = input("Enter a string: ")
2
3 # Using Python's slicing for maximum efficiency
4 reversed_string = user_input[::-1]
5
6 print(f"Reversed: {reversed_string}")
```

The right pane shows a terminal window with the following output:

```
PS C:\Users\hp\OneDrive\Desktop\ai> & "c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:\Users\hp\.vscode\extensions\ms-python.python-2025.18.0-win32-x64\bundled\lib\site-packages\debugpy\launcher" "50075" -- "c:\Users\hp\OneDrive\Desktop\ai\task1.py"
Enter a string: 40 50 60 70
Reversed: 70 60 50 40
PS C:\Users\hp\OneDrive\Desktop\ai>
```

### Task 3: Modular Design Using AI Assistance (String Reversal Using Functions)

❖ **Scenario**

The string reversal logic is needed in multiple parts of an application.

❖ **Task Description**

Use GitHub Copilot to generate a function-based Python program that:

- Uses a user-defined function to reverse a string
- Returns the reversed string
- Includes meaningful comments (AI-assisted)

❖ **Expected Output**

- Correct function-based implementation
- Screenshots documenting Copilot's function generation

## ➤ Sample test cases and outputs

```
task1.py •
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
1 def reverse_string_functional(text):
2     """
3         Reverses the input string and returns it.
4     """
5     reversed_text = ""
6     for char in text:
7         |    reversed_text = char + reversed_text
8     return reversed_text
9
10 # Testing the function
11 input_str = input("Enter text: ")
12 result = reverse_string_functional(input_str)
13 print(f"Result: {result}")

PS C:\Users\hp\OneDrive\Desktop\ai> ^C
PS C:\Users\hp\OneDrive\Desktop\ai>
PS C:\Users\hp\OneDrive\Desktop\ai> c:\od 'c:\Users\hp\OneDrive\Desktop\ai' & 'c:\Users\hp\Applata\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\.vscode\extensions\ms-python
on/debugger-2025.18.0-win32-x64\bundled\libs\debugger\launcher' '53825' ... 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter text: Hello
Result: olleH
PS C:\Users\hp\OneDrive\Desktop\ai>
```

## Task 4: Comparative Analysis - Procedural vs Modular Approach (With vs Without Functions)

### ❖ Scenario

You are asked to justify design choices during a code review.

### ❖ Task Description

Compare the Copilot-generated programs:

#### ➤ Without functions (Task 1)

#### ➤ With functions (Task 3) Analyze them based on:

➤ Code clarity

➤ Reusability

➤ Debugging ease

➤ Suitability for large-scale applications

### ❖ Expected Output

Comparison table or short analytical report

Feature	Procedural (Without Functions)	Modular (With Functions)
Code Clarity	Easy for tiny scripts; messy for large ones.	Very high; logic is isolated and named.
Reusability	Must copy-paste code to use it again.	Can be called anywhere in the app.
Debugging	<del>Harder to isolate where an error occurs.</del> Not suitable for large	<del>Easy to unit test the specific function.</del> Essential for professional
Scalability	applications.	development.

### Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to String Reversal)

#### ❖ Scenario

Your mentor wants to evaluate how AI handles alternative logic paths.

#### ❖ Task Description

Prompt GitHub Copilot to generate:

- A loop-based string reversal approach
- A built-in / slicing-based string reversal approach

#### ❖ Expected Output

- Two correct implementations

#### ➢ Comparison discussing:

- Execution flow
- Time complexity
- Performance for large inputs
- When each approach is appropriate.

The screenshot shows the VS Code interface with the code editor open to a file named 'task1.py'. The code implements two functions for reversing strings: 'reverse\_iterative' using a loop and 'reverse\_slicing' using Python's slice operator. It also includes a test input section. The Python Debugger interface is visible at the top right, showing a dropdown menu for 'Python Debugger: Cu...'. The code editor has dark-themed syntax highlighting.

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
1 def reverse_iterative(input_string):
2     reversed_str = ""
3     for char in input_string:
4         reversed_str = char + reversed_str
5     return reversed_str
6
7 def reverse_slicing(input_string):
8     return input_string[::-1]
9
10 test_input = input("Enter a string: ")
11
12 print(reverse_iterative(test_input))
13 print(reverse_slicing(test_input))
```

The screenshot shows the VS Code terminal window. The user runs the command 'python task1.py' to execute the script. The terminal output shows the program prompting for input ('Enter a string:'), receiving the input '1 2 3 4 5', and then printing the reversed strings from both methods. The output is as follows:

```
s:\debgpy\launcher" "50436" "--" "c:\Users\hp\OneDrive\Desktop\ai\task1.py"
PS C:\Users\hp\OneDrive\Desktop\ai>
PS C:\Users\hp\OneDrive\Desktop\ai> cd "c:\Users\hp\OneDrive\Desktop\ai"; & "c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "c:\Users\hp\.vscode\extensions\ms-python.on_debugger-2025.18.0-win32-x64\bundled\libs\debgpy\launcher" "57517" "--" "c:\Users\hp\OneDrive\Desktop\ai\task1.py"
Enter a string: 1 2 3 4 5
5 4 3 2 1
5 4 3 2 1
PS C:\Users\hp\OneDrive\Desktop\ai>
```