AI Assistant Coding

Assignment -1

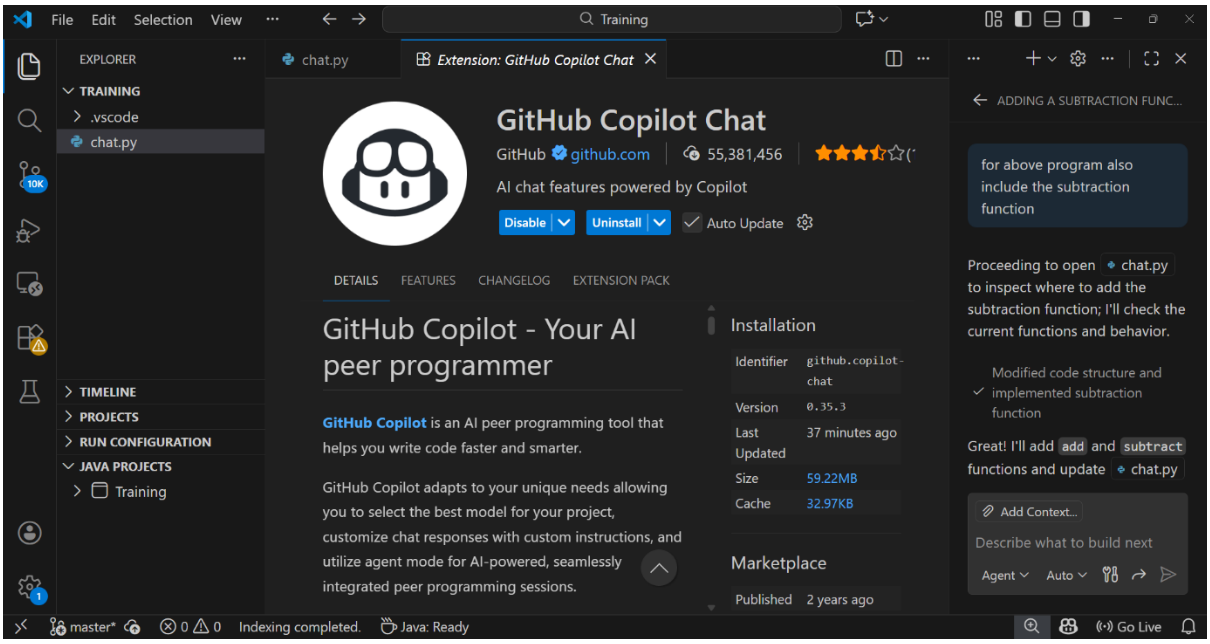
Name: T. Swetha

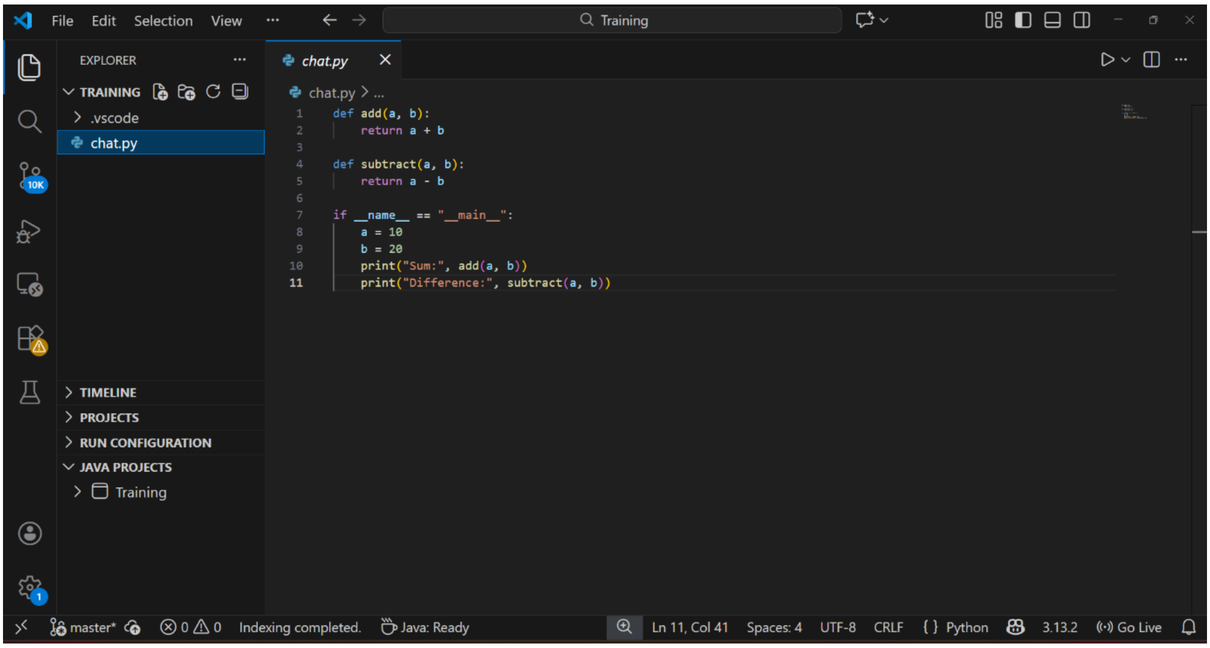
Roll no:2303a51317

Batch:05

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

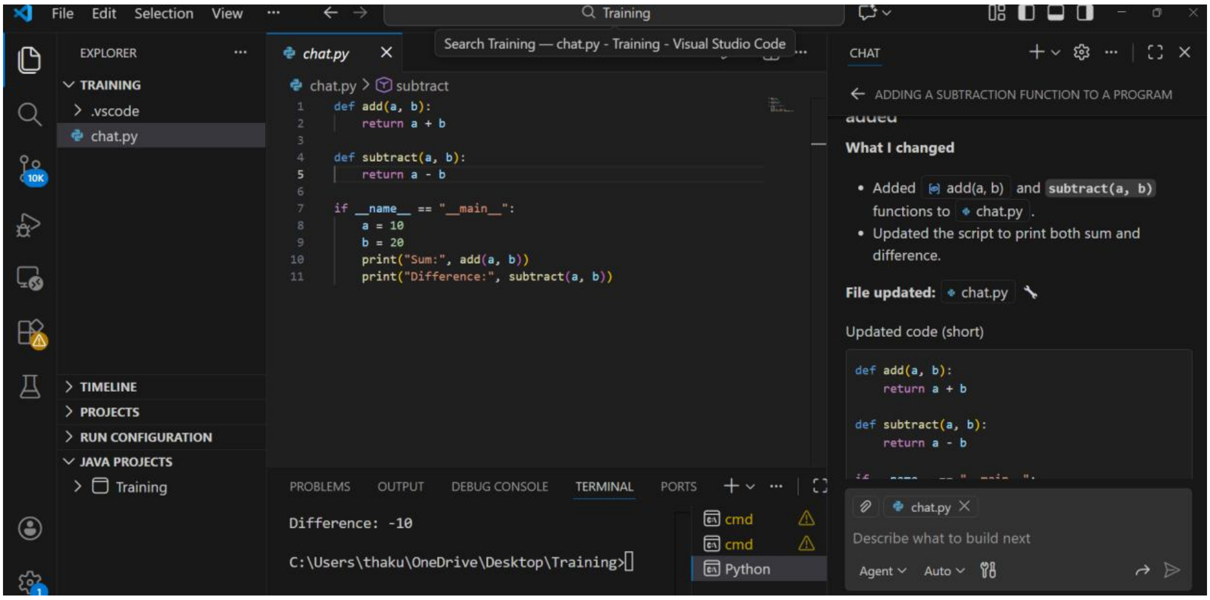

File Edit Selection View
Training
EXTENSIONS: MA...
python
Python Debug... 151ms
Python Debugger extensi...
Microsoft
Python 377ms
Python language support...
Microsoft
Pylance 1085ms
A performant, feature-ric...
Microsoft
Python Environ... 91ms
Provides a unified python...
Microsoft
Python Indent 16.3M
Correct Python indentation
Kevin Rose
Install
Python Exten... 12.8M
Popular Visual Studio Co...
Don Jayamanne
Install
Extension: Python
Python
Microsoft microsoft.com 198,472,333 (620)
Python language support with extension access points for IntelliSense (Pylance), Debugging...
Disable
Uninstall
Switch to Pre-Release Version
Auto Update
DETAILS FEATURES CHANGELOG EXTENSION PACK
Python extension for Visual Studio Code
A Visual Studio Code extension with rich support for the Python language (for all actively supported Python versions), providing access points for extensions to seamlessly integrate and offer support for IntelliSense (Pylance), debugging (Python Debugger), formatting, linting, code navigation, refactoring, variable explorer, test explorer, environment management (NEW Python Environments Extension).
Support for vscode.dev
The Python extension does offer some support when running on vscode.dev (which
Installation
Identifier ms-python.python
Version 2025.20.1
Last Updated 42 minutes ago
Size 29.61MB
Cache 651B
Marketplace
Published 9 years ago
Last 17 hours ago
master* 0 0 Indexing completed. Java: Ready
Go Live


File Edit Selection View
Training
EXPLORER
TRAINING
.vscode
chat.py
TIMELINE
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
Training
chat.py
Extension: GitHub Copilot Chat
GitHub Copilot Chat
GitHub github.com 55,381,456
AI chat features powered by Copilot
Disable
Uninstall
Auto Update
DETAILS FEATURES CHANGELOG EXTENSION PACK
GitHub Copilot - Your AI peer programmer
GitHub Copilot is an AI peer programming tool that helps you write code faster and smarter.
GitHub Copilot adapts to your unique needs allowing you to select the best model for your project, customize chat responses with custom instructions, and utilize agent mode for AI-powered, seamlessly integrated peer programming sessions.
Installation
Identifier github.copilot-chat
Version 0.35.3
Last Updated 37 minutes ago
Size 59.22MB
Cache 32.97KB
Marketplace
Published 2 years ago
ADDING A SUBTRACTION FUNC...
for above program also include the subtraction function
Proceeding to open chat.py to inspect where to add the subtraction function; I'll check the current functions and behavior.
Modified code structure and implemented subtraction function
Great! I'll add add and subtract functions and update chat.py
Add Context...
Describe what to build next
Agent Auto
master* 0 0 Indexing completed. Java: Ready
Go Live


File Edit Selection View
Training
EXPLORER
TRAINING
.vscode
chat.py
chat.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

if __name__ == "__main__":
    a = 10
    b = 20
    print("Sum:", add(a, b))
    print("Difference:", subtract(a, b))
TIMELINE
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
Training
master* 0 0 Indexing completed. Java: Ready Ln 11, Col 41 Spaces: 4 UTF-8 CRLF { } Python 3.13.2 Go Live

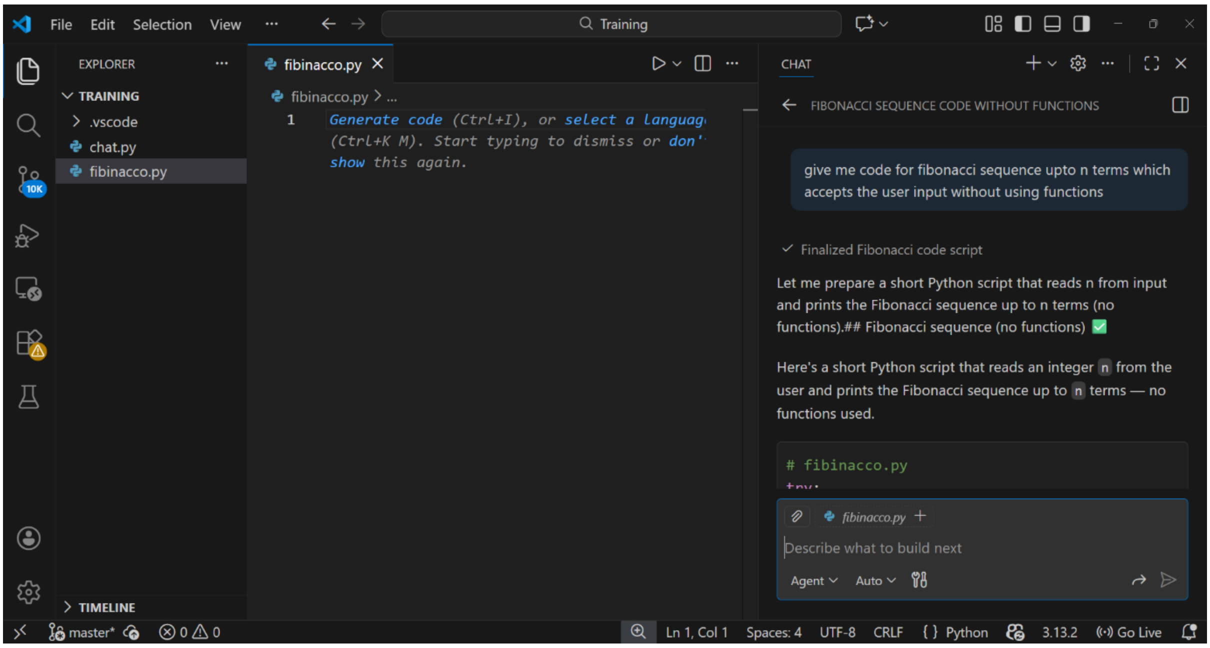

File Edit Selection View
Training
Search Training — chat.py - Training - Visual Studio Code
EXPLORER
TRAINING
.vscode
chat.py
chat.py
subtract
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

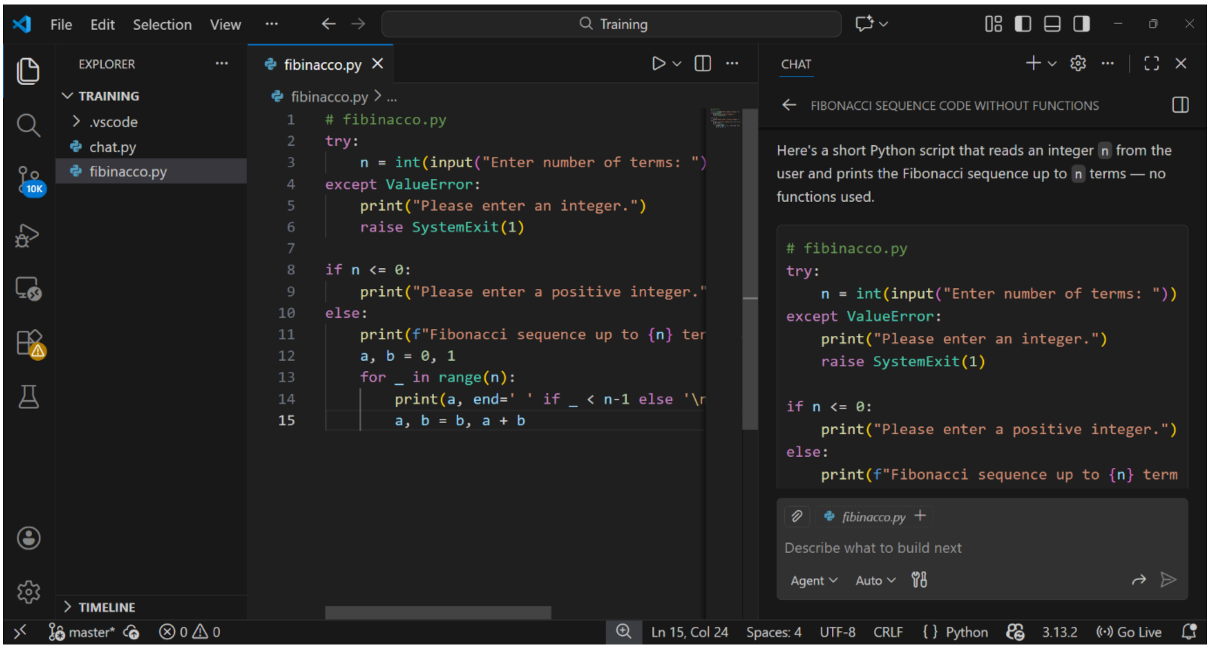
if __name__ == "__main__":
    a = 10
    b = 20
    print("Sum:", add(a, b))
    print("Difference:", subtract(a, b))
TIMELINE
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
Training
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Difference: -10
C:\Users\thaku\OneDrive\Desktop\Training>
cmd
cmd
Python
CHAT
ADDING A SUBTRACTION FUNCTION TO A PROGRAM
What I changed
Added add(a, b) and subtract(a, b) functions to chat.py .
Updated the script to print both sum and difference.
File updated: chat.py
Updated code (short)
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
chat.py
Describe what to build next
Agent Auto

Task 1: AI-Generated Logic Without Modularization (Fibonacci Sequence Without Functions)

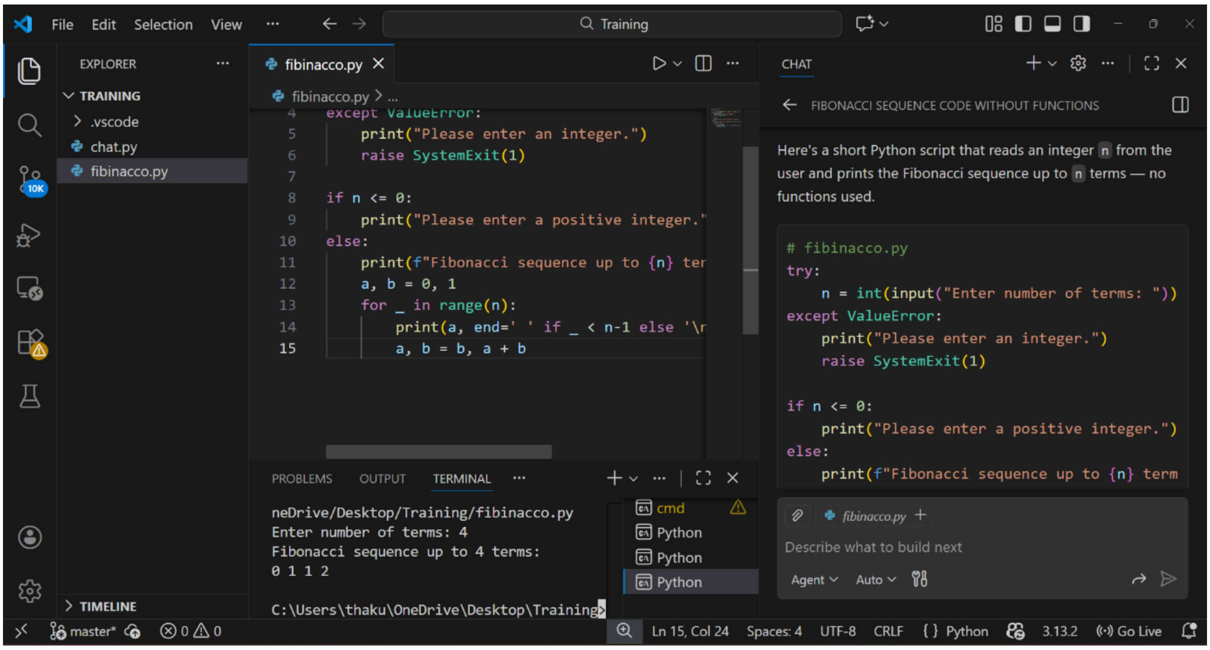
Step 1: Giving prompt: Give me code for fibonacci sequence up to n terms which accepts the user input without using functions



Step 2: Code



Step 3: Output



Step 4: Explanation of Code

Input Handling:

- Uses try-except to ensure the user enters an integer.
- Exits gracefully if input is invalid.

Validation:

- Checks if n <= 0.
- Prompts user to enter a positive integer if invalid.

Output Message:

- Prints a header with correct pluralization (term vs terms).

Initialization:

- Starts with a = 0, b = 1 (first two Fibonacci numbers).

Loop Logic:

- Runs for _ in range(n) to generate n terms.
- Prints current value of a.
- Updates values with a, b = b, a + b.

Printing Style:

- Numbers are printed on the same line separated by spaces.
- Last number ends with a newline.

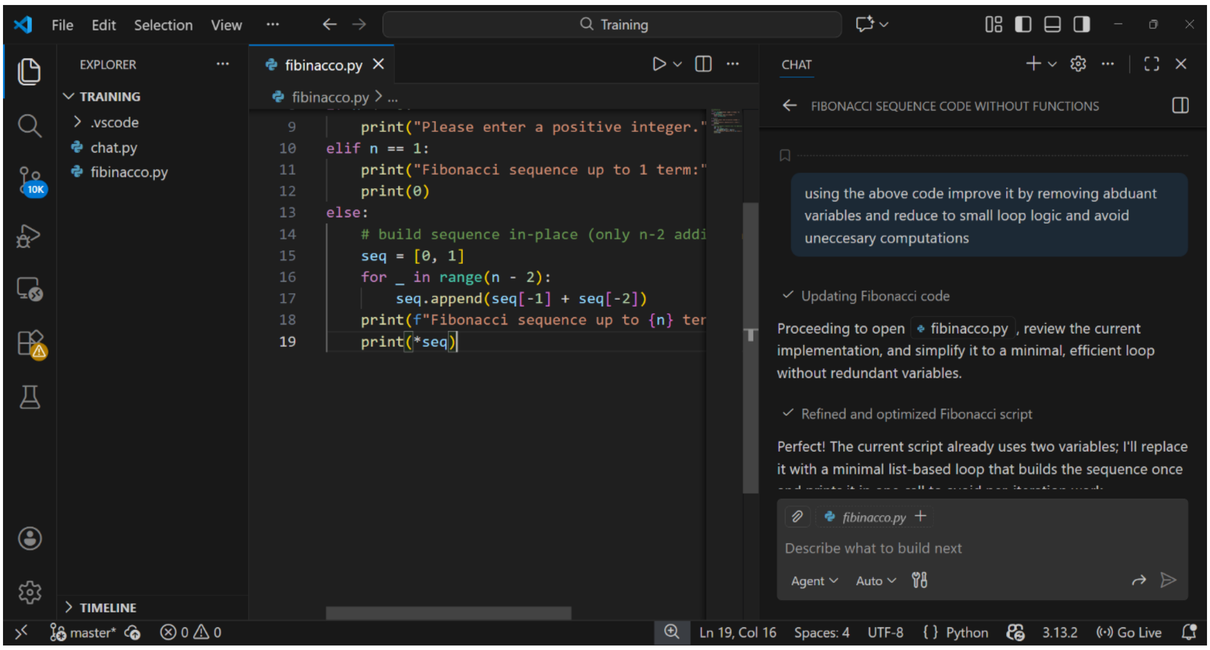
Result:

- Produces the Fibonacci sequence up to n terms.

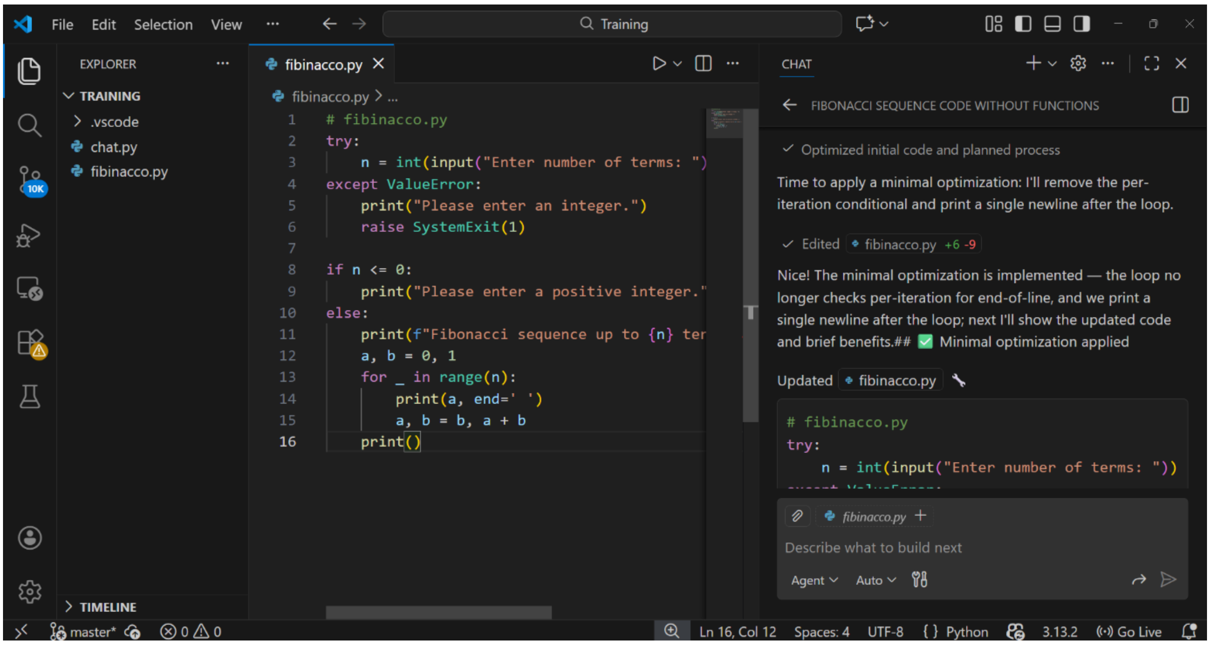
Example:
Input: 4 → Output: 0 1 1 2

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

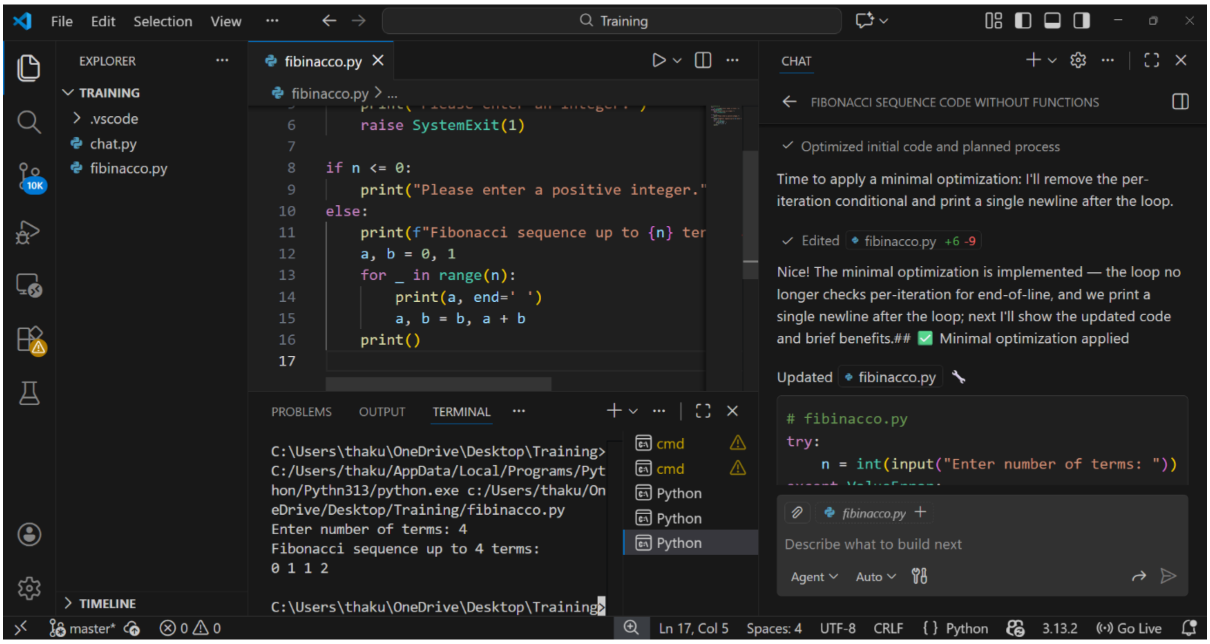
Step 1: Giving Prompt: Using the above code improve it by removing abundant variables and reduce to small loop logic and avoid unnecessary computations / optimize the code



Step 2: Code



Step 3: Output



Step 4: Code Explanation

Input Handling

- Uses try-except to ensure the user enters an integer.
- Exits safely if input is invalid.

Validation

- Checks if n <= 0.
- Prompts user to enter a positive integer if invalid.

Output Message

- Prints a header with correct pluralization (term vs terms).

Initialization

- Starts with a = 0, b = 1 (first two Fibonacci numbers).

Loop Logic

- Runs for _ in range(n) to generate n terms.
- Prints current value of a.
- Updates values with a, b = b, a + b.

What Was Inefficient in the Original Code

The original code had stuff that wasn't needed to generate the Fibonacci sequence.

A volatile memory variable was used to hold the sum and added extra length.

It's because you're putting the fibonacci sequence into a list as well a program like this will just output the result, but store it in memory somewhere.

The loop was constructed in more than one step, which made the code less readable

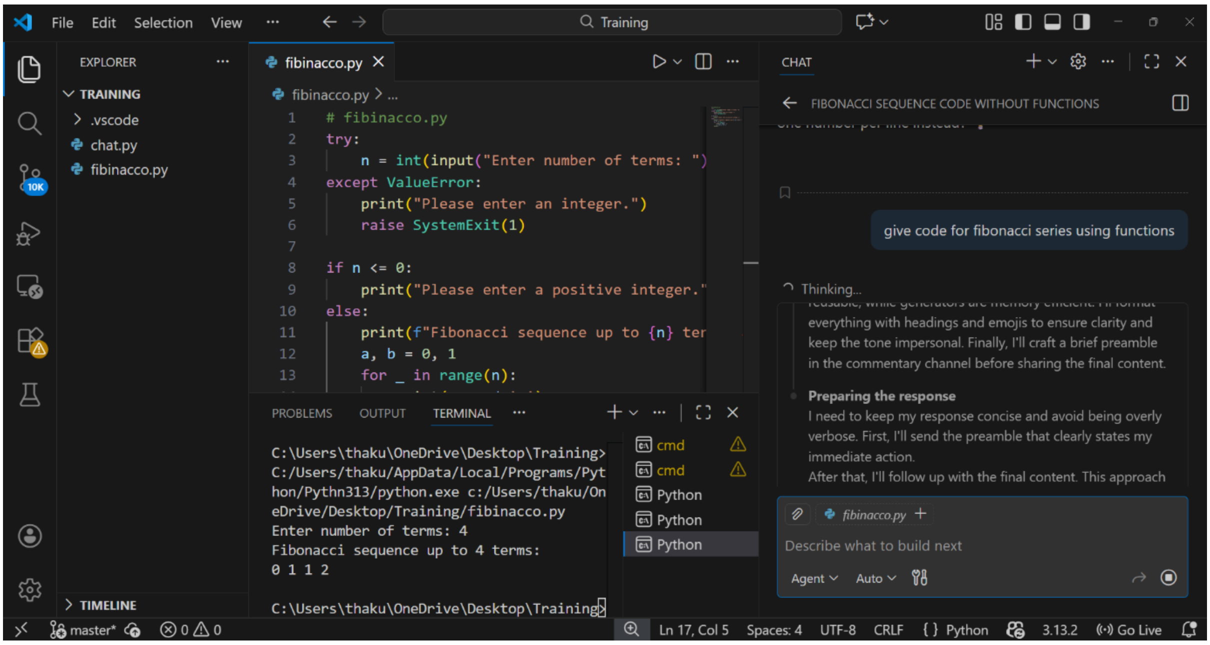
How the optimized/python version brings performance improvement and improved readability.

Unnecessary variables were discarded and the values were updated by tuple assignment in an efficient manner.

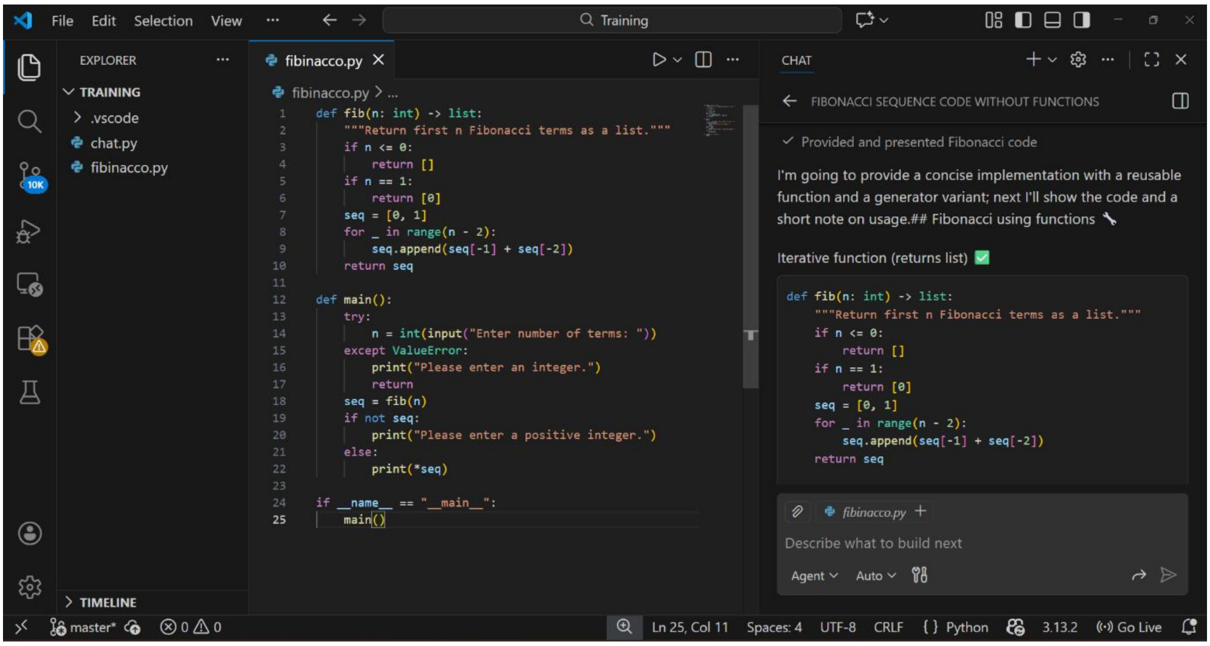
Your code also writes to disc, I assume you want to speed the codeup, so this is not good. Your optimized version that this saved memory.3) that it prints the number instead of storing a whole vector with all solutions.

Task 3: Modular Design Using AI Assistance (Fibonacci Using Functions)

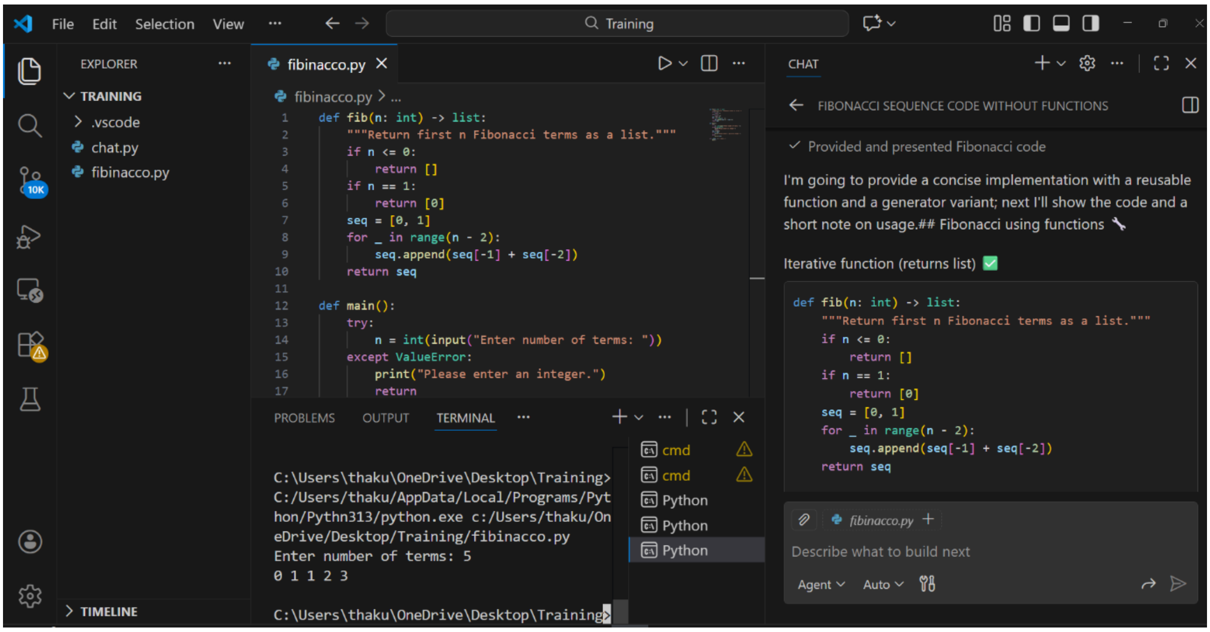
Step 1: Prompt: Give code for Fibonacci sequence using functions.



Step 2: Code:



Step 3: Output



Step 4: Explanation of code

Function Design

- fib(n) cleanly separates Fibonacci logic from input/output.
- Returns a list of the first n Fibonacci terms.

Validation

- If n <= 0: returns an empty list.
- If n == 1: returns [0].
- Handles edge cases gracefully.

Sequence Generation

- Starts with [0, 1].
- Uses a loop for _ in range(n - 2) to generate remaining terms.
- Each new term = sum of last two (seq[-1] + seq[-2]).

Main Function

- Prompts user for input.
- Uses try-except to catch invalid input (ValueError).
- Calls fib(n) to generate sequence.
- Prints a warning if input is invalid or non-positive.

Output

- Prints the sequence neatly with print(*seq) (space-separated values).

Practices

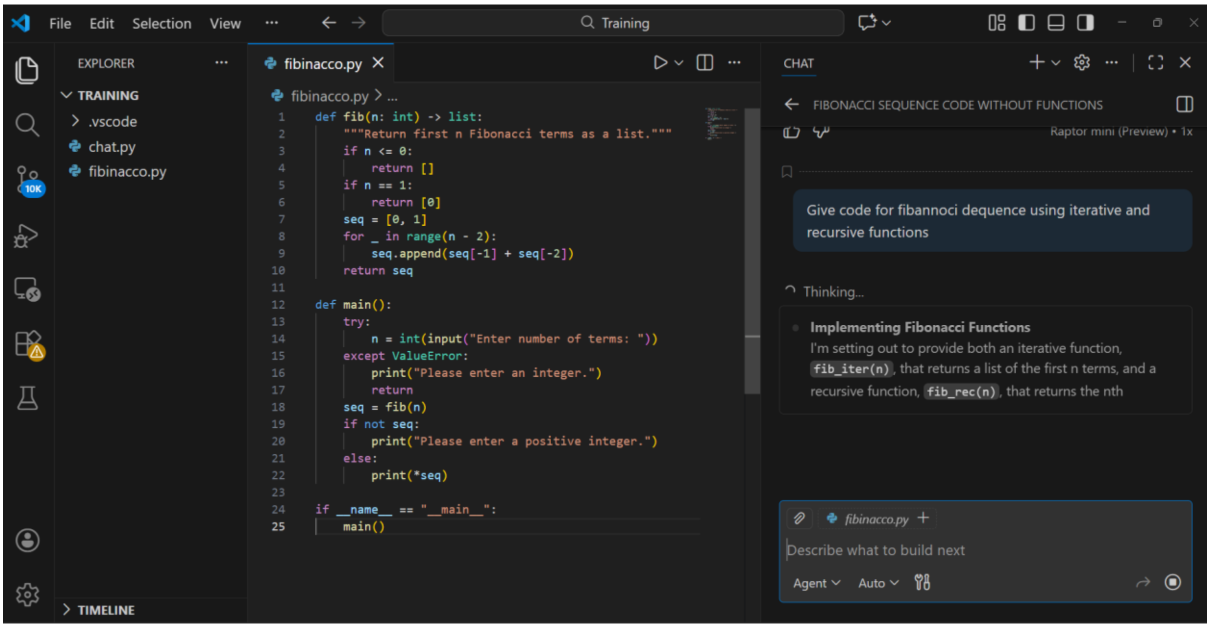
- Includes a docstring for clarity.
- Uses __name__ == "__main__" guard for modularity.
- Keeps logic reusable and testable.

Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

Feature	**Without Functions**	**With Functions**
Code Clarity	Logic is written in one block; harder to read when long	Logic is separated into a named function → easier to understand
Reusability	Cannot reuse Fibonacci logic without rewriting	Can call the function anywhere in program
Debugging Ease	Bugs must be traced in main logic, mixed with other code	Errors isolated in function → easier to test & fix
Suitability for Larger Systems	Poor; not scalable, becomes messy with added features	Good; fits into bigger systems, easier to maintain
Testing	Hard to unit test a part of code independently	Function can be tested separately with multiple inputs
Maintainability	Low; changes affect entire code block	High; changes only in function, no impact on main flow
Performance Impact	No function call overhead (very small benefit)	Minimal overhead but worth it for structure & scaling

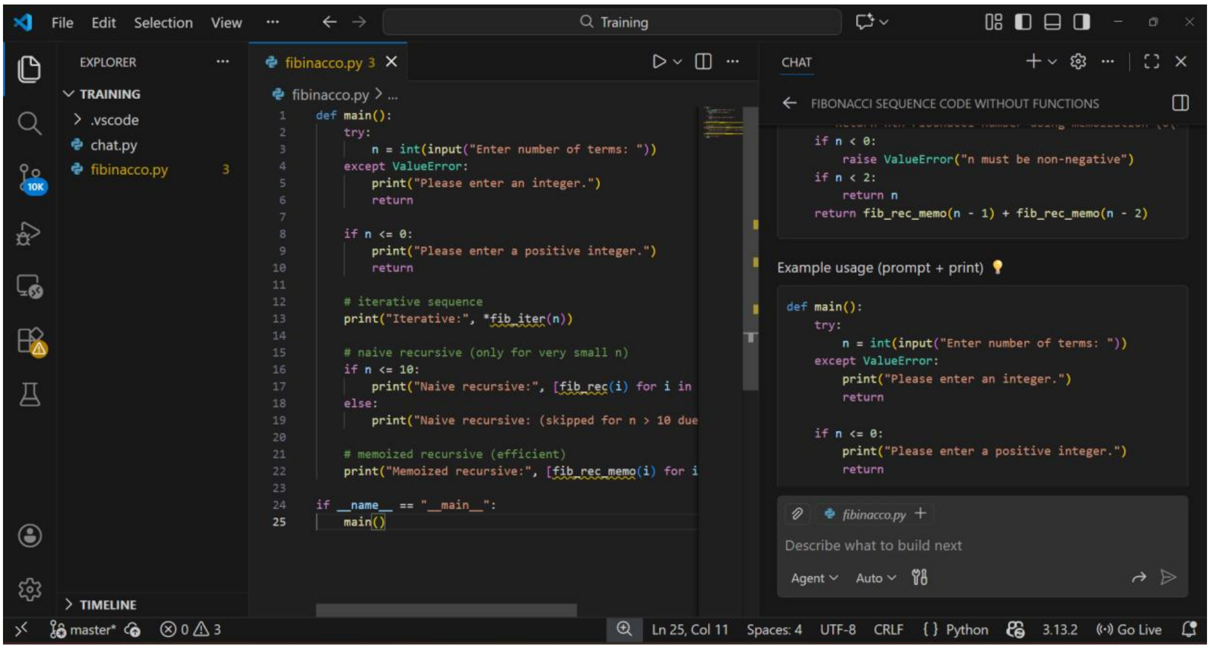
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches for Fibonacci Series)

Step 1: Prompt: Give code for fibanocci series using iterative and recursive functions

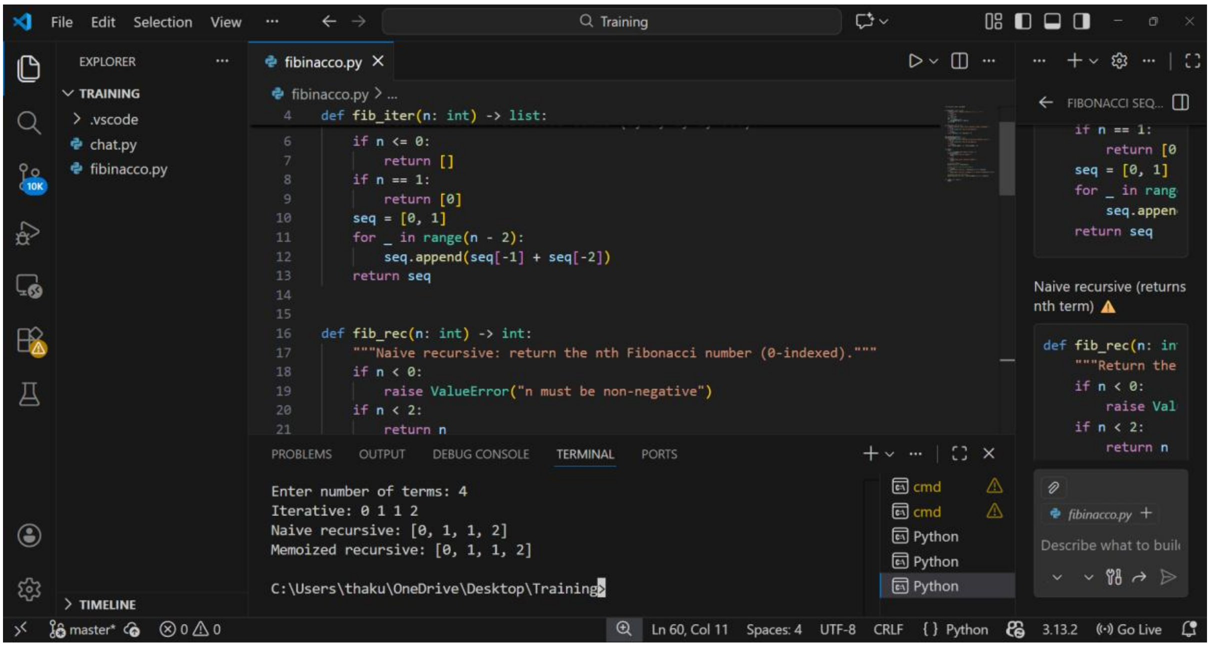


Step 2: Code

Iterative:



Step 3: Output



Step 4: Explanation of code

fib_iter(n) → Iterative list builder, efficient O(n).

- fib_rec(n) → Naive recursion, exponential cost, only practical for very small n.
- fib_rec_memo(n) → Recursive with @lru_cache, efficient O(n) via memoization.
- Input Handling → Validates integer input, rejects negatives/zero.
- Main Function →
  - Prints iterative sequence always.
  - Prints naive recursive only if n ≤ 10.
  - Prints memoized recursive for any n.
- Good Practices → Clear separation of methods, input validation, modular design, efficient memoization.

Example (n=6):
Iterative: 0 1 1 2 3 5
Naive recursive: [0, 1, 1, 2, 3, 5]
Memoized recursive: [0, 1, 1, 2, 3, 5]

Comparison covering:

Aspect	Iterative	Recursive
Time Complexity	O(n)	$O(2^n)$ (very slow due to repeated calls)
Space Complexity	O(1)	O(n) (stack memory for calls)
Performance for Large n	Excellent (can handle 10^7+ if needed)	Poor (fib(50) may take seconds/minutes)
Memory Usage	Very low	High because of recursion stack
Scalability	Best for real systems	Not scalable without optimization
Risk	No crash risk	Stack Overflow for large n