

AI Assisted Coding

Assignment Number:1.3

Name:Chirra Jyothika

HtNo:2303A51280

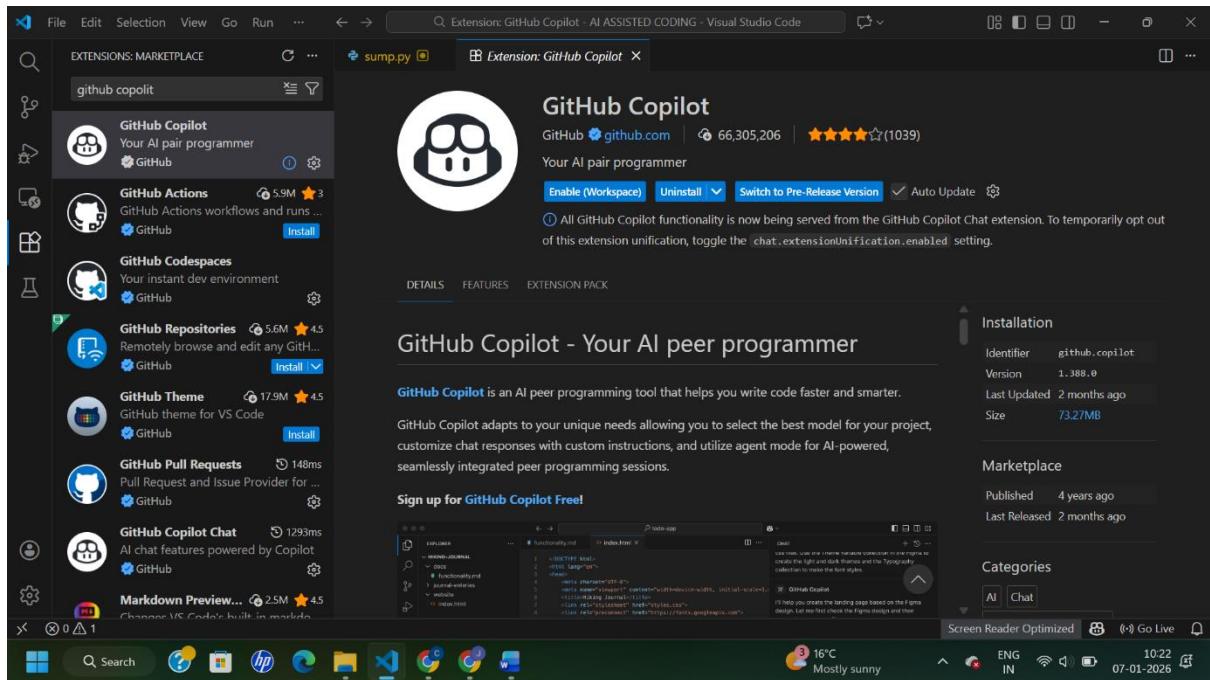
BtNo:05

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

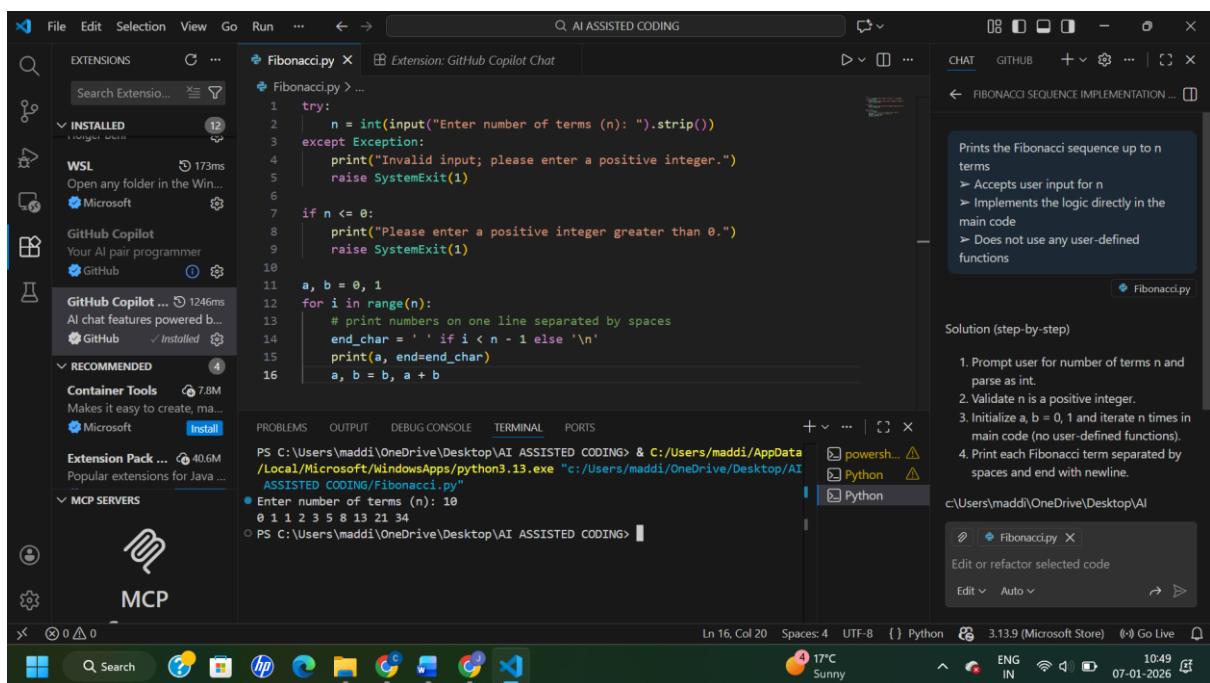
Task 0:

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





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



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

optimize this fibonacci code to remove redundant variables

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

- File Explorer:** Shows extensions like GitHub Copilot Chat (installed), WSL, GitHub, Container Tools, and MCP Servers.
- Code Editor:** Displays the original Fibonacci code:

```
1 try:
2     n = int(input("Enter number of terms (n): ").strip())
3 except Exception:
4     print("Invalid input; please enter a positive integer.")
5     raise SystemExit(1)
6
7 if n <= 0:
8     print("Please enter a positive integer greater than 0.")
9     raise SystemExit(1)
10
11 if n == 1:
12     print(0)
13 else:
14     fibs = [0, 1]
15     while len(fibs) < n:
16         fibs.append(fibs[-1] + fibs[-2])
17     print(' '.join(map(str, fibs[:n])))
```
- Terminal:** Shows the output of running the script: "Enter number of terms (n): 10" followed by the sequence "0 1 2 3 5 8 13 21 34".
- Chat Panel:** A message from GitHub Copilot Chat: "optimize this fibonacci code to remove redundant variables".
- Solution Panel:** Step-by-step instructions:
 - Read and validate n as before.
 - Build Fibonacci sequence in a single list (no a/b or end_char).
 - Print the first n terms by joining the list into a space-separated string.
- Status Bar:** Shows the file is saved (+7 -6), the Python extension is installed, and the system status (24°C, Sunny, ENG IN, 11:33, 07-01-2026).

simplify variable usage and make the fibonacci loop more readable

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

- File Explorer:** Shows extensions like GitHub Copilot Chat (installed), WSL, GitHub, Container Tools, and MCP Servers.
- Code Editor:** Displays the simplified Fibonacci code:

```
1 try:
2     n = int(input("Enter number of terms (n): ").strip())
3 except Exception:
4     print("Invalid input; please enter a positive integer.")
5     raise SystemExit(1)
6
7 if n <= 0:
8     print("Please enter a positive integer greater than 0.")
9     raise SystemExit(1)
10
11 fibs = [0]
12 if n > 1:
13     fibs.append(1)
14     for _ in range(2, n):
15         fibs.append(fibs[-1] + fibs[-2])
16     print(' '.join(map(str, fibs[:n])))
```
- Terminal:** Shows the output of running the script: "Enter number of terms (n): 10" followed by the sequence "0 1 2 3 5 8 13 21 34".
- Chat Panel:** A message from GitHub Copilot Chat: "simplify variable usage and make the fibonacci loop more readable".
- Solution Panel:** Step-by-step instructions:
 - Validate input n as before.
 - Initialize fibs with required first terms using minimal variables.
 - Use a clear for-loop to build remaining terms and print once.
- Status Bar:** Shows the file is saved (+6 -7), the Python extension is installed, and the system status (24°C, Sunny, ENG IN, 11:35, 07-01-2026).

optimize this code to avoid unnecessary computations and memory usage

The screenshot shows the Microsoft Visual Studio Code interface. The left sidebar displays extensions like GitHub Copilot and Container T... under the RECOMMENDED section. The main editor window contains Python code for a Fibonacci sequence:

```
1 try:
2     n = int(input("Enter number of terms (n): ").strip())
3 except Exception:
4     print("Invalid input; please enter a positive integer.")
5     raise SystemExit(1)
6
7 if n <= 0:
8     print("Please enter a positive integer greater than 0.")
9     raise SystemExit(1)
10
11 # replace the list-based Fibonacci construction with a constant-space loop
12 if n == 1:
13     print(0)
14 else:
15     prev, curr = 0, 1
16     print(prev, end=' ')
17     for i in range(1, n):
18         end_char = ' ' if i < n - 1 else '\n'
19         print(curr, end=end_char)
20         prev, curr = curr, prev + curr
```

The terminal below shows the output of running the script:

```
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CODING/Fibonacci.py"
Enter number of terms (n): 10
0 1 1 2 3 5 8 13 21 34
PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>
```

The status bar at the bottom indicates it's a GPT-5 mini session.

refactor this fibonacci program to improve performance and readability

The screenshot shows the Microsoft Visual Studio Code interface. The right panel displays a task card titled "refactor this fibonacci program to improve performance and readability".

The code in the editor has been modified to use a single loop and handle errors using ValueError instead of SystemExit:

```
1 import sys
2
3 try:
4     n = int(input("Enter number of terms (n): ").strip())
5 except ValueError:
6     print("Invalid input; please enter a positive integer.")
7     sys.exit(1)
8
9 if n <= 0:
10     print("Please enter a positive integer greater than 0.")
11     sys.exit(1)
12
13 prev, curr = 0, 1
14 for i in range(n):
15     end = ' ' if i < n - 1 else '\n'
16     print(prev, end=end)
17     prev, curr = curr, prev + curr
```

The terminal output remains the same as in the previous screenshot.

#Optimize this Fibonacci code by removing redundant variables, simplifying the loop logic, and avoiding unnecessary computations.

```

1  try:
2      n = int(input("Enter number of terms (n): ").strip())
3  except ValueError:
4      print("Invalid input; please enter a positive integer.")
5      raise SystemExit(1)
6
7  if n <= 0:
8      print("Please enter a positive integer greater than 0.")
9      raise SystemExit(1)
10
11 prev, curr = 0, 1
12 for i in range(n):
13     print(prev, end=' ' if i < n - 1 else '\n')
14     prev, curr = curr, prev + curr

```

Solution (step-by-step)

1. Remove the unused sys import and use SystemExit for exits.
2. Inline the print end selection to remove the temporary end variable.
3. Keep the constant-space loop for performance.

c:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING\Fibonacci.py

Apply input validation simplification and inline printing.

Edited Fibonacci.py +3 -6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

powerhell... Python Python

PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING> Enter number of terms (n): 10
0 1 1 2 3 5 8 13 21 34

PS C:\Users\maddi\OneDrive\Desktop\AI ASSISTED CODING>

Ln 14, Col 35 Spaces: 4 UTF-8 { } Python 3.13.9 (Microsoft Store) 11:38 07-01-2026 ENG IN 24°C Sunny

What Was Inefficient in the Original Code

- The original code used **extra variables** that were not necessary to generate the Fibonacci sequence.
- A **temporary variable** was used to store the sum of numbers, which increased code length.
- The program stored the Fibonacci sequence in a **list**, even though the requirement was only to display the output.
- The loop logic was written in **multiple steps**, making the code less readable

How the Optimized Version Improves Performance and Readability

- Redundant variables were removed, and **tuple assignment** was used to update values efficiently.
- The optimized code prints values directly instead of storing them, which **reduces memory usage**.
- The loop logic was simplified, making the code **shorter and easier to understand**.

- Overall performance was improved by using **constant memory** and cleaner logic.

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

```

# Function-based Fibonacci generator with simple CLI and AI-assisted
# documentation.

def generate_fibonacci(n):
    """Return the first n Fibonacci numbers as a list.

    Uses an iterative approach that appends each new term based on
    the previous two. Returns an empty list for n <= 0.

    """
    if n <= 0:
        return []
    seq = [0]
    if n == 1:
        return seq
    seq.append(1)
    # Iteratively build remaining terms using last two values.
    for _ in range(2, n):
        seq.append(seq[-1] + seq[-2])
    return seq

try:
    n = int(input("Enter number of terms (n): ").strip())

```

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)

An iterative Fibonacci implementation

```

# Function-based Fibonacci generator with simple CLI and AI-assisted coding
def generate_fibonacci(n):
    """Return the first n Fibonacci numbers as a list.

    Uses an iterative approach that appends each new term based on the last two. Returns an empty list for n <= 0.
    """
    if n <= 0:
        return []
    seq = [0]
    if n == 1:
        return seq
    seq.append(1)
    # Iteratively build remaining terms using last two values.
    for _ in range(2, n):
        seq.append(seq[-1] + seq[-2])
    return seq

def fib(n):
    """Return the nth Fibonacci number using an iterative approach"""
    return generate_fibonacci(n)[-1]

```

AI ASSISTED CODING

ITERATIVE FIBONACCI IMPLEMENTATION AND CLI UPDATES

Completed: Inspect Fibonacci.py (1/3)

Update: Implemented iterative fib(n) and moved the CLI into a main guard.

File: fibonacci.py — added fib(n) (iterative, 0-based) and preserved generate_fibonacci(n).

How to run:

```
python Fibonacci.py
```

Enter a non-negative integer when prompted; the script prints the first n terms and fib(n-1) if n>0.

Would you like me to add unit tests or change the CLI to accept command-line args?

Todos (3/3)

Fibonacci.py +

Describe what to build next

Agent Auto

A recursive Fibonacci implementation

```

# Function-based Fibonacci generator with simple CLI and AI-assisted coding
def generate_fibonacci(n):
    """Return the first n Fibonacci numbers as a list.

    Uses an iterative approach that appends each new term based on the last two. Returns an empty list for n <= 0.
    """
    if n <= 0:
        return []
    seq = [0]
    if n == 1:
        return seq
    seq.append(1)
    # Iteratively build remaining terms using last two values.
    for _ in range(2, n):
        seq.append(seq[-1] + seq[-2])
    return seq

def fib(n):
    """Return the nth Fibonacci number using an iterative approach"""
    return generate_fibonacci(n)[-1]

```

AI ASSISTED CODING

ADDING RECURSIVE FIBONACCI FUNCTION AND TESTING OPTIONS

A recursive Fibonacci implementation

Verified Fibonacci implementation and updated progress

Running the script with input 10 to verify output now.

```
echo 10 | & C:/Users/maddi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/maddi/OneDrive/Desktop/AI ASSISTED CODING/Fibonacci.py"
```

Completed: Run and verify (3/3)

Verified — script output for input 10:

- Sequence: 0 1 1 2 3 5 8 13 21 34
- Printed value: fib(9) = 34

What next: add a small unit test calling fib_recursive() or update the CLI to show fib_recursive(n-1) as well. Want me to add tests or a demo?

Fibonacci.py +

Describe what to build next

Agent Auto

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	StackOverflow for large n