# AI Assisted Coding

# Assignment - 01

**Name : A.Rithika**

**Roll No : 2303A54048**

**Batch : 48**

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

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

❖ **Scenario :** You are asked to write a quick numerical sequence generator for a learning platform prototype.

- **Prompt used :**
  # Write a Python program to generate the Fibonacci sequence up to n terms # Take user input for n # Implement the logic directly in the main program # Do not use any user-defined functions # Print the Fibonacci sequence.
- **Generated code :**

```python
try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit


if n <= 0:
    print("Please enter a positive integer")
else:
    prev, curr = 0, 1
    print(f"Fibonacci sequence up to {n} terms:")
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(prev, end=end)
        prev, curr = curr, prev + curr
```

- **Sample Input :**
  Enter no of terms : 5
- **Sample Output :**
  Fibonacci sequence up to 5 terms:
  0 1 1 2 3
- **Short Explanation of Logic :**

The program first takes an integer **n** as input from the user, which represents the number of Fibonacci terms to be printed. It initializes the first two Fibonacci numbers as **0** and **1**. Using a **loop**, the program prints each term and updates the values by adding the previous two numbers to generate the next term in the sequence. The entire logic is written directly in the main program without using any user-defined functions.

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

❖ **Scenario :The prototype will be shared with other developers and needs optimization.**

- **Prompt used :**
  # Optimize this Fibonacci code # Simplify variable usage # Improve readability and efficiency.

- **Generated code :**

```python
try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit


if n <= 0:
    print("Please enter a positive integer")
else:
    prev, curr = 0, 1
    print(f"Fibonacci sequence up to {n} terms:")
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(prev, end=end)
        prev, curr = curr, prev + curr
```

- **Sample Input :**
  Enter the number of terms: 5

- **Sample Output :**
  Fibonacci sequence up to 5 terms:
  0 1 1 2 3

- **Short Explanation of Logic :**
  The optimized program improves the original Fibonacci code by removing unnecessary variables and simplifying the loop logic. It generates each Fibonacci number using only the previous two values, avoiding redundant calculations. This makes the code more readable, efficient, and easier to understand while producing the same correct output.

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

❖ **Scenario :** The Fibonacci logic is now required in multiple modules of an application.
- **Prompt used :**

   # Write a Python program using a function to generate Fibonacci sequence # The function should take n as input # Include meaningful comments

- **Generated code :**

```
"""
task3.py

Generate Fibonacci sequence using a function.

The function `fibonacci(n)` returns a list containing the first `n`
Fibonacci numbers (starting from 0). The main code reads `n` from
standard input, validates it, and prints the sequence.
"""

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

    Args:
    n (int): Number of terms to generate. If n <= 0, an empty list
    is returned.

    Returns:
    list[int]: List of Fibonacci numbers of length n.
    """
    if n <= 0:
    return []
    if n == 1:
    return [0]

    seq = [0, 1]
    while len(seq) < n:
    seq.append(seq[-1] + seq[-2])
    return seq


if __name__ == "__main__":
    try:
    n = int(input("Enter the number of terms: "))
    except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

    if n <= 0:
    print("Please enter a positive integer")
    else:
    # Generate and print the sequence on one line, space-separated
    seq = fibonacci(n)
    print(f"Fibonacci sequence up to {n} terms:")
    print(" ".join(str(x) for x in seq))
```

- **Sample Input :**

Enter the number of terms: 5
- **Sample Output :**

Fibonacci sequence up to 5 terms:
   0 1 1 2 3

- **Short Explanation of Logic :**

  The program defines a user-defined function that generates the Fibonacci sequence up to n terms. The main program takes input from the user and calls this function. Inside the function, the Fibonacci numbers are calculated using a loop and printed (or returned). Using a function makes the code modular, reusable, and easier to maintain.

## Task 4: Comparative Analysis – Procedural vs Modular Fibonacci Code

- ❖ **Scenario :** You are participating in a code review session.
  - **Prompt used :**
  - **Prompt 1: Basic Comparison**

Compare a Fibonacci program written without functions and a Fibonacci program written using functions.
Analyze the comparison in terms of:
- Code clarity
- Reusability
- Ease of debugging
- Suitability for larger systems
Provide the answer in simple academic language.

- **Prompt 2: Table Format**

Create a comparison table between procedural Fibonacci code (without functions) and modular Fibonacci code (with functions). Include parameters such as code clarity, reusability, debugging ease, and scalability.

- **Prompt 3: Short Analytical Explanation**

  Write a short analytical report comparing procedural and modular Fibonacci implementations. Explain which approach is better and why. Limit the explanation to 5–6 lines.

- **Sample Output :**

  **Comparison (Fibonacci Programs)**

  Code clarity: Functions: named blocks (e.g., fib(n)) separate intent from implementation, so the main flow is concise and easy to read; No functions: everything is inline, which can be fine for very short scripts but becomes harder to follow as logic grows.
  Reusability: Functions: easily reused, imported, and called with different arguments; No functions: logic is tied to the script and must be copied or refactored to reuse.

Ease of debugging: Functions: let you test and isolate small units (unit tests, step-through debugging) so locating bugs is simpler; No functions: debugging requires examining the whole linear flow and is more error-prone.

Suitability for larger systems: Functions: enable modular design, easier maintenance, and integration into larger codebases; No functions: poor scalability—hard to extend, test, or integrate into services or libraries.

**Procedural vs Modular Fibonacci — short analytical report :**

Procedural implementations keep logic inline, making very small scripts straightforward but quickly cluttered as logic grows.

Modular implementations encapsulate Fibonacci generation in named functions, which improves clarity and documents intent.

Functions greatly increase reusability: the same routine can be imported and reused without duplication.

Debugging is easier with functions since units can be tested in isolation and traced independently.

For larger systems, modular design scales far better—it's maintainable, testable, and integrates cleanly into other components.

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

❖ **Scenario : Your mentor wants to assess AI's understanding of different algorithmic paradigms.**

- **Prompt used :**

**Iterative Prompt :**

# Generate Fibonacci sequence using iterative approach.

**Recursive Prompt**

# Generate Fibonacci sequence using recursive approach.

- **Generated code :**

```python
"""Iterative Fibonacci sequence generator.

This script reads an integer `n` from standard input and prints the
first `n` Fibonacci numbers using an iterative loop (no recursion).
"""


try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit


if n <= 0:
    print("Please enter a positive integer")
```

```python
else:
    # Initialize the first two Fibonacci numbers
    a, b = 0, 1
    # Print the sequence iteratively, printing values on the fly
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(a, end=end)
        a, b = b, a + b
```

```python
"""Recursive Fibonacci sequence generator.

This script defines a recursive function `fib(k)` that returns the k-th
Fibonacci number and then uses it to print the first `n` Fibonacci numbers.
Note: naive recursion has exponential time complexity; use small `n`.
"""


def fib(k):
    """Return the k-th Fibonacci number using recursion.

    Base cases: fib(0) = 0, fib(1) = 1.
    Recursive case: fib(k) = fib(k-1) + fib(k-2).
    """
    if k <= 1:
        return k
    return fib(k - 1) + fib(k - 2)



try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

if n <= 0:
    print("Please enter a positive integer")
else:
    # Print the first n Fibonacci numbers using the recursive function
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(fib(i), end=end)
```

- **Sample Input :**

Enter the number of terms: 5
- **Sample Output :**
    0 1 1 2 3

EXPLORER — LAB1.3 — task1.py

Welcome | task1.py

```python
try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

if n <= 0:
    print("Please enter a positive integer")
else:
    a, b = 0, 1
    seq = []
    for _ in range(n):
        seq.append(str(a))
        a, b = b, a + b
    print(f"Fibonacci sequence up to {n} terms:")
    print(" ".join(seq))
```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS          Python

```
anumandlarithika@Mac lab1.3 % 5
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab1.3/task1.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 %
```

CHAT — FIBONACCI SEQUENCE PROGRAM IN PYTHON

Write a Python program to print Fibonacci sequence up to n terms

Take user input for n

Do not use any user-defined functions

Implement logic directly in main code

✓ Crafted Fibonacci program and managed code

I'll add a short TODO entry, then update task1.py with the Fibonacci program.

✓ Created 1 todo

Now I'll read the current task1.py so I can update it safely.

✓ Read task1.py

I'm about to write the Fibonacci code into task1.py. I'll apply the patch now.

✓ Edited task1.py +17 -1

Done — task1.py updated with the Fibonacci program.

To run it:

```
python3 task1.py
```

Want me to run it here with a sample input to verify output?

GPT-5 mini · 1x

task1.py +

Describe what to build next

Agent ∨  GPT-5 mini ∨

Ln 17, Col 1   Spaces: 4   UTF-8   Python   3.9.6

---

EXPLORER — LAB1.3 — task1.py, task2.py

task2.py | task1.py

```python
try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

if n <= 0:
    print("Please enter a positive integer")
else:
    prev, curr = 0, 1
    print(f"Fibonacci sequence up to {n} terms:")
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(prev, end=end)
        prev, curr = curr, prev + curr
```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS          Python

```
anumandlarithika@Mac lab1.3 % 5
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab1.3/task1.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab1.3/task2.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 % 10
zsh: command not found: 10
anumandlarithika@Mac lab1.3 %
```

CHAT — FIBONACCI SEQUENCE PROGRAM IN PYTHON

Optimize this Fibonacci code

Simplify variable usage

Improve readability and efficiency

✓ Optimized Fibonacci sequence code

I'll add an in-progress TODO, read task1.py, then patch it to a simpler, in-place print loop and mark the TODO completed.

✓ Managed and tracked todo items

✓ Edited task1.py +5 -6

✓ Completed: Optimize Fibonacci code (2/2)

Progress: optimized task1.py — simplified variables and print-on-the-fly; TODO updated.

What I changed:

- File: task1.py — renamed variables to prev, curr, removed list allocation, and print numbers directly with proper spacing.

Run it:

```
python3 task1.py
```

Want me to run it here with a sample input to verify output?

task2.py +

Describe what to build next

Agent ∨  GPT-5 mini ∨

Ln 15, Col 39   Spaces: 4   UTF-8   LF   Python   3.9.6

## Comparison Table

| Aspect | Procedural (no functions) | Modular (with functions) |
|---|---|---|
| Code clarity | Clear for very short, one-off scripts; inline logic becomes harder to follow as size grows. | High: named functions separate intent from implementation, making the main flow concise and readable. |
| Reusability | Low: logic is tied to the script and must be copied to reuse. | High: functions can be imported and reused across programs. |
| Ease of debugging | Lower: harder to isolate faults; debugging requires tracing the whole linear flow. | Higher: small units can be tested and debugged independently (unit tests, isolated tracing). |
| Suitability for larger systems (scalability) | Poor: difficult to maintain, extend, or integrate into larger codebases. | Excellent: supports modular design, testing, and easier maintenance in large systems. |

## Top screenshot

EXPLORER

> LAB1.3
- task1.py
- task2.py
- task3.py
- task4.py
- task5.py
- task6.py

Tabs: task3.py | task4.py | task5.py | task6.py ×

task6.py > ...

```python
"""Recursive Fibonacci sequence generator.

This script defines a recursive function `fib(k)` that returns th
Fibonacci number and then uses it to print the first `n` Fibonacc
Note: naive recursion has exponential time complexity; use small
"""

def fib(k):
    """Return the k-th Fibonacci number using recursion.

    Base cases: fib(0) = 0, fib(1) = 1.
    Recursive case: fib(k) = fib(k-1) + fib(k-2).
    """
    if k <= 1:
        return k
    return fib(k - 1) + fib(k - 2)


try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

if n <= 0:
    print("Please enter a positive integer")
else:
    # Print the first n Fibonacci numbers using the recursive fun
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(fib(i), end=end)
```
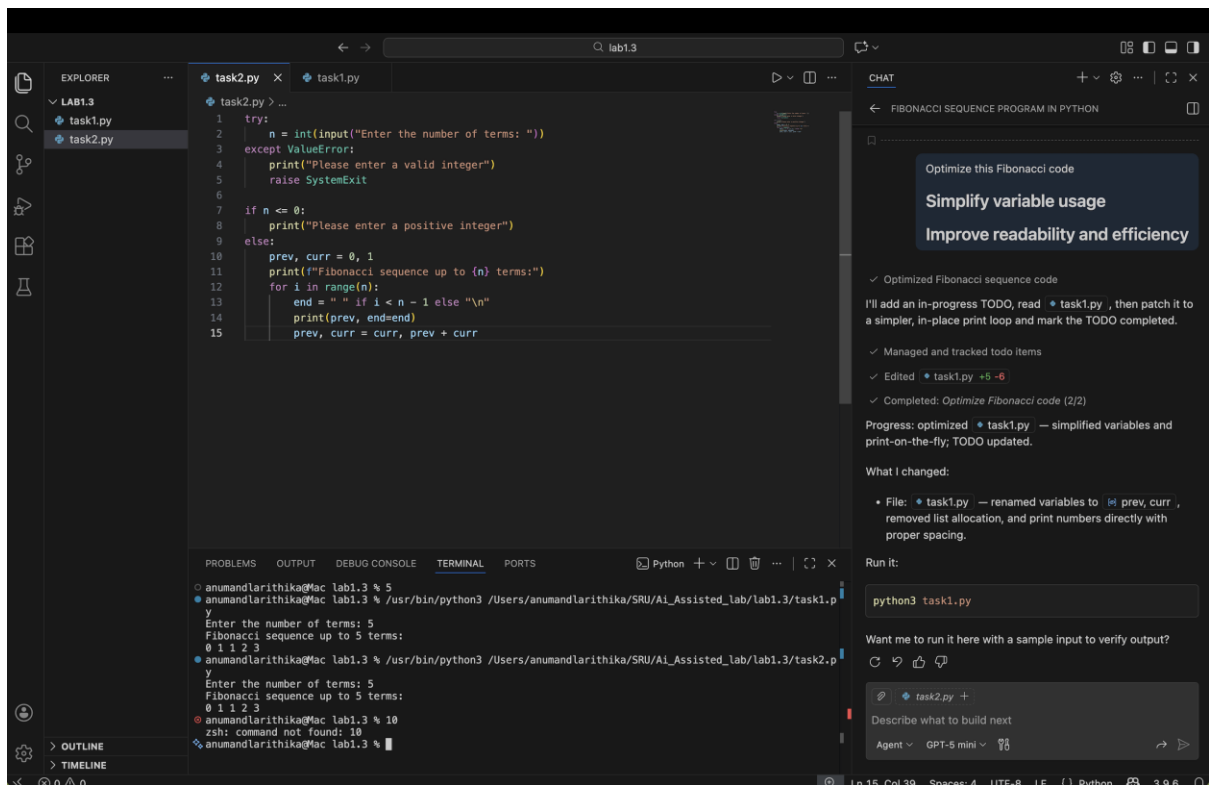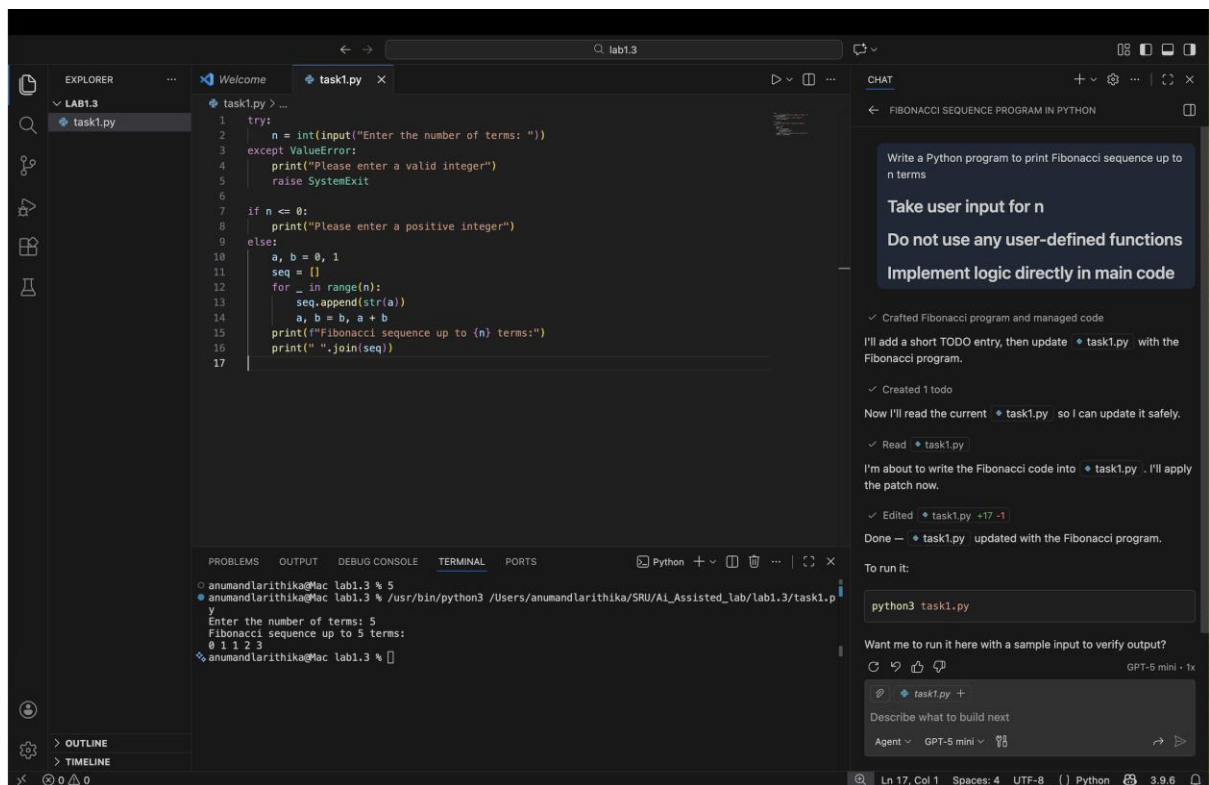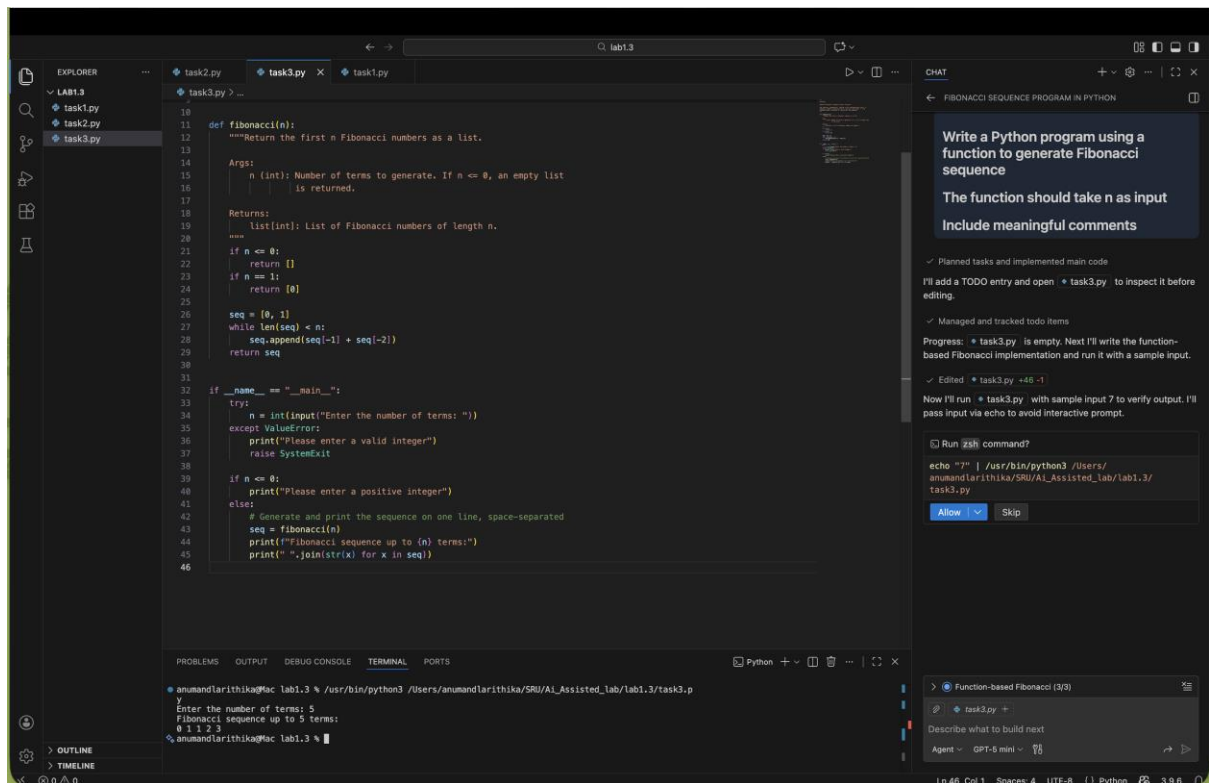
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

```
anumandlarithika@Mac lab1.3 % echo 7 | /usr/bin/python3 /Users/a
numandlarithika/SRU/Ai_Assisted_lab/lab1.3/task6.py
Enter the number of terms: 0 1 1 2 3 5 8
anumandlarithika@Mac lab1.3 %
```

Python
zsh

CHAT

← FIBONACCI SEQUENCE PROGRAM IN PYTHON

### Generate Fibonacci sequence using recursive approach

✓ Finalized Fibonacci implementation and organized tool calls

I'll add a TODO entry, create `task6.py` with a recursive Fibonacci implementation, and run it with a sample input to verify output.

✓ Created 5 todos

✓ Edited ● task6.py +32 -1

✓ ● echo 7 | /usr/bin/python3
/Users/anumandlarithika/SRU/Ai_Assisted_lab/lab1.3/task6.py

✓ Completed: Recursive Fibonacci (5/5)

Done — ● task6.py created and tested with sample input (output shown).

Next steps: I can add memoization to `fib()` for better performance or integrate this into a module — which would you prefer?

> Todos (5/5)
⊘ ● task6.py +

Describe what to build next

Agent ∨  GPT-5 mini ∨

Ln 1, Col 1  Spaces: 4  UTF-8  LF  {} Python  3.9.6

## Bottom screenshot

EXPLORER

> LAB1.3
- task1.py
- task2.py
- task3.py
- task4.py
- task5.py

Tabs: task2.py | task3.py | task4.py | task5.py ×

task5.py > ...

```python
"""Iterative Fibonacci sequence generator.

This script reads an integer `n` from standard input and prints t
first `n` Fibonacci numbers using an iterative loop (no recursion
"""

try:
    n = int(input("Enter the number of terms: "))
except ValueError:
    print("Please enter a valid integer")
    raise SystemExit

if n <= 0:
    print("Please enter a positive integer")
else:
    # Initialize the first two Fibonacci numbers
    a, b = 0, 1
    # Print the sequence iteratively, printing values on the fly
    for i in range(n):
        end = " " if i < n - 1 else "\n"
        print(a, end=end)
        a, b = b, a + b
```

PROBLEMS  OUTPUT  TERMINAL

```
anumandlarithika@Mac lab1.3 % 10
zsh: command not found: 10
anumandlarithika@Mac lab1.3 % clear
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Ass
isted_lab/lab1.3/task3.p
y
Enter the number of terms: 5
Fibonacci sequence up to 5 terms:
0 1 1 2 3
anumandlarithika@Mac lab1.3 % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Ass
isted_lab/lab1.3
/task5.py
Enter the number of terms: 5
0 1 1 2 3
anumandlarithika@Mac lab1.3 %
```

Python

CHAT

← FIBONACCI SEQUENCE PROGRAM IN PYTHON

integrates cleanly into other components.

### Generate Fibonacci sequence using iterative approach

✓ Crafted iterative Fibonacci sequence code

I'll add a TODO entry and create/update ● task5.py with an iterative Fibonacci implementation.

✓ Managed and tracked todo items for planning

✓ Edited ● task5.py +23 -1

✓ Completed: Iterative Fibonacci (4/4)

Progress: Added ● task5.py with an iterative Fibonacci generator and updated the TODOs.

To run:

```
python3 task5.py
```

Want me to run it here with a sample input to show output?

> Todos (4/4)
⊘ ● task5.py +

Describe what to build next

Agent ∨  GPT-5 mini ∨

Ln 23, Col 1  Spaces: 4  UTF-8  {} Python  3.9.6