G.Tejomayi

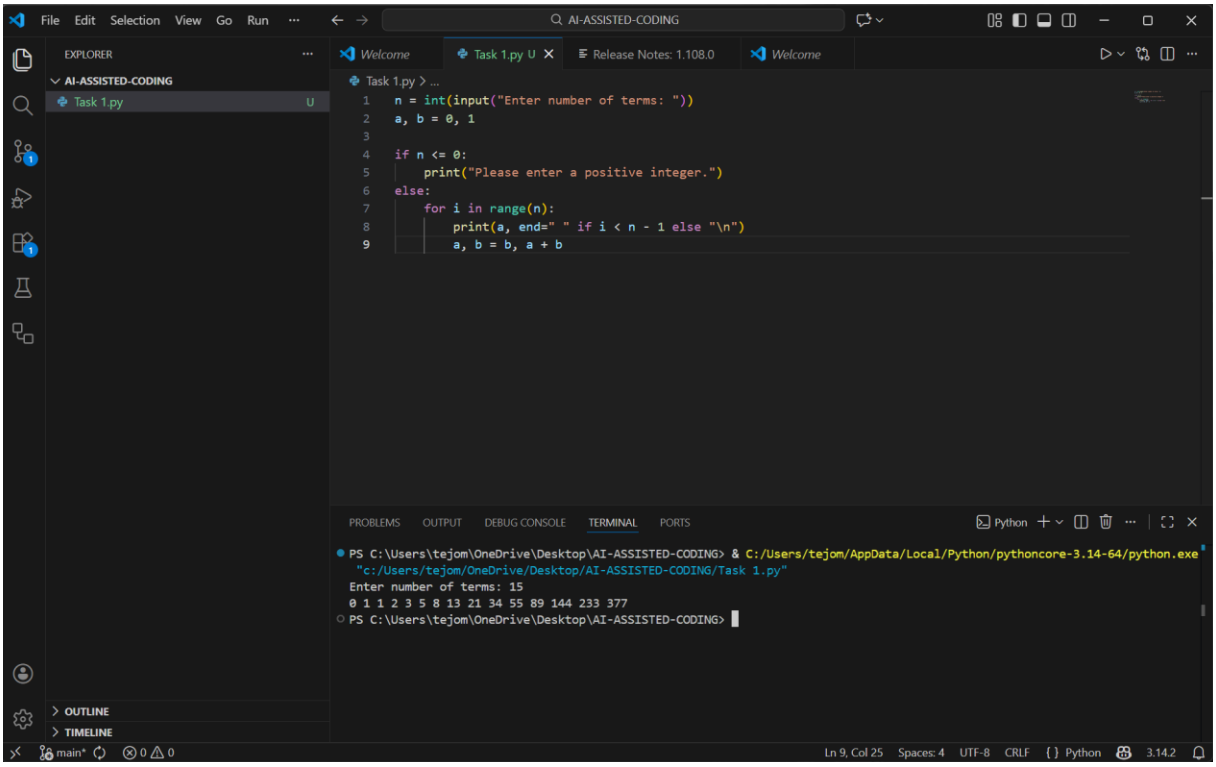
2303A54064 B-47B

Lab 1

Assignment Number 1.3

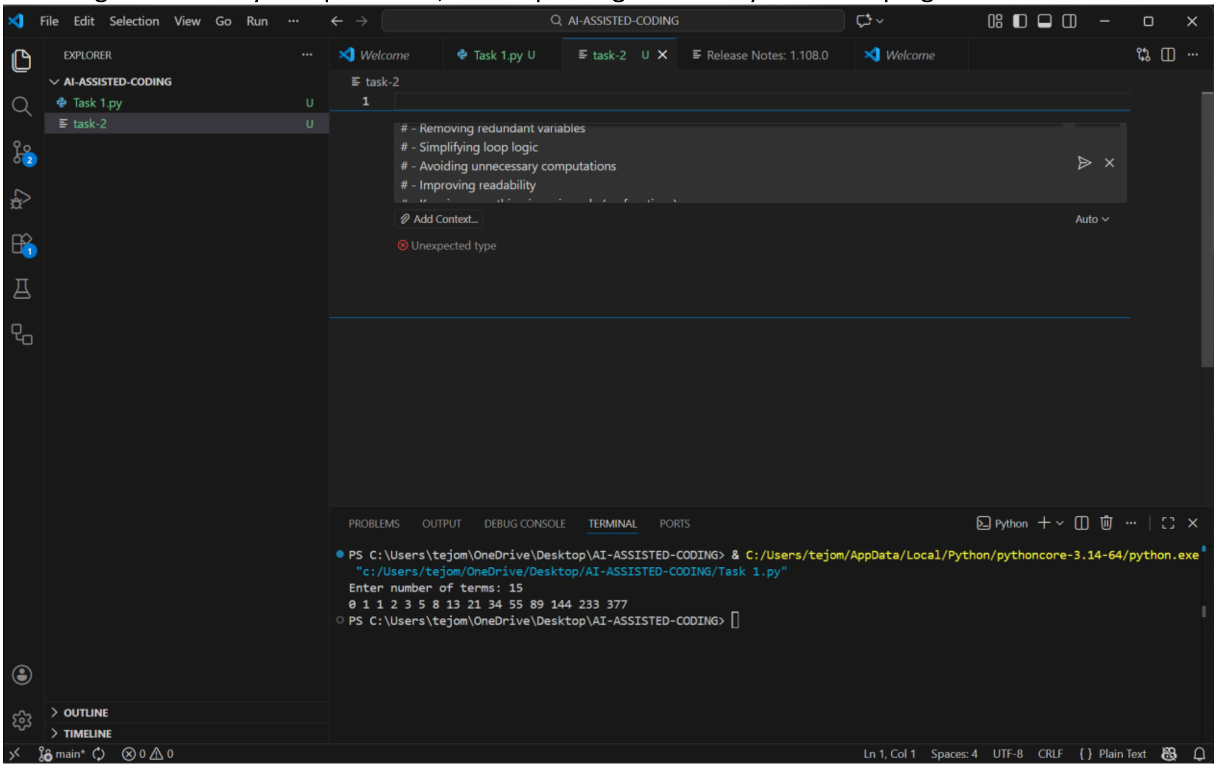
TASK:1

PROMPT: Write a Python program that takes user input n and prints the Fibonacci sequence up to n terms using main code only without any user-defined functions



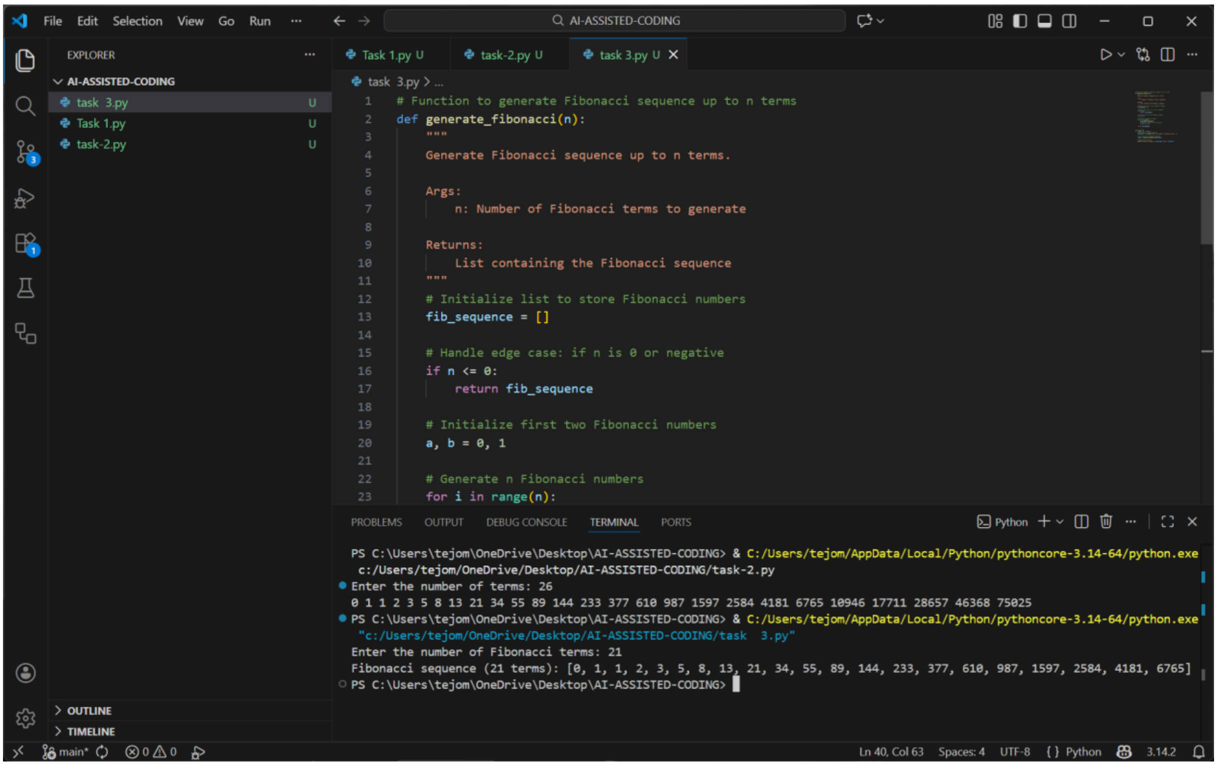
TASK:2

PROMPT: Optimize this Fibonacci code by removing redundant variables, simplifying loop logic, avoiding unnecessary computations, and improving readability while keeping it function-free



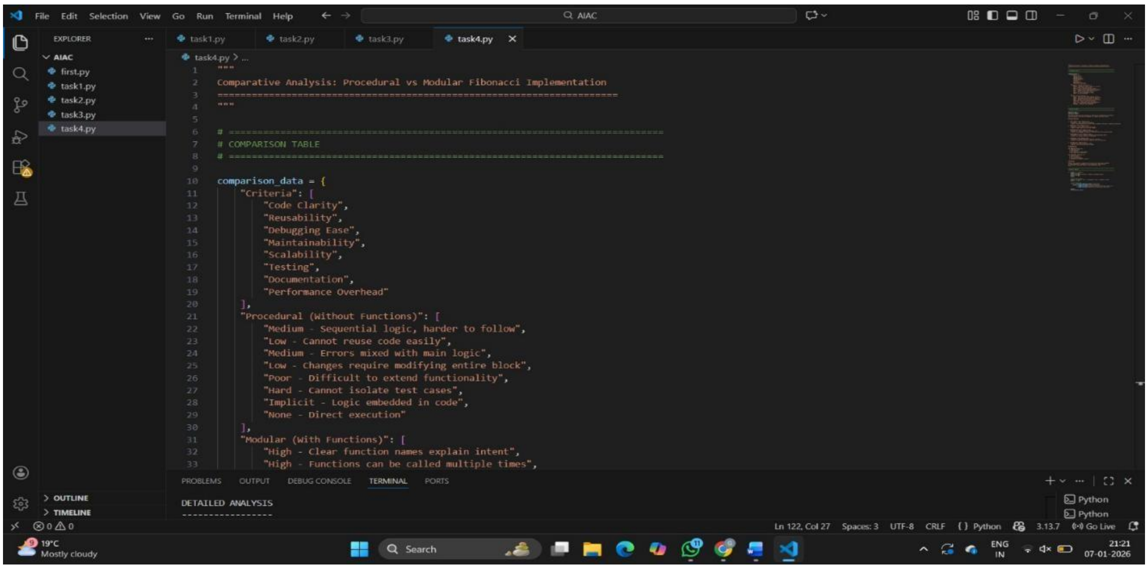
TASK:3

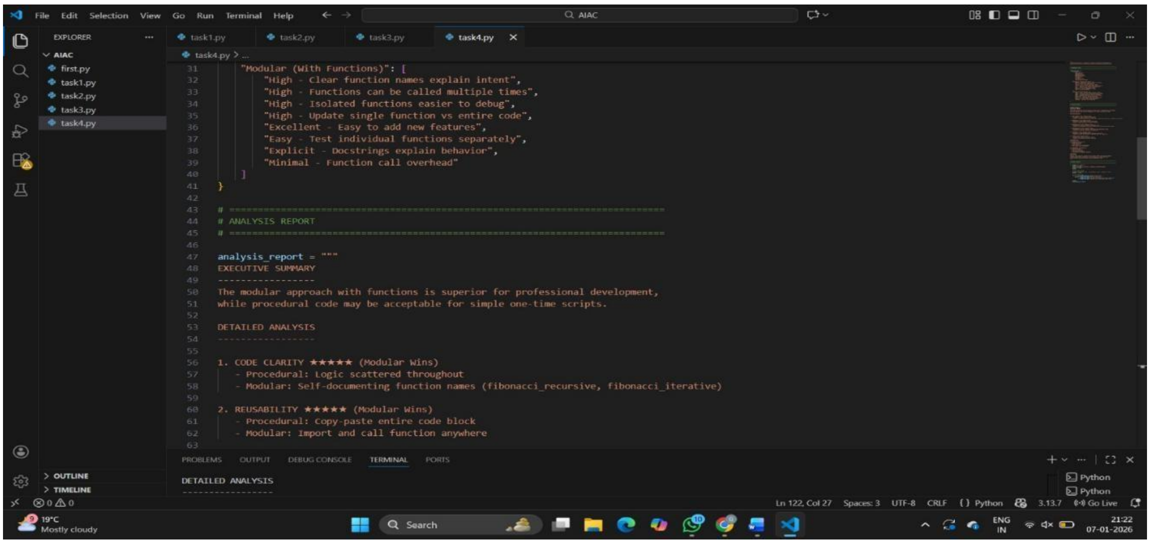
PROMPT: Write a Python program that uses a user-defined function to generate and return the Fibonacci sequence up to n terms with clear and meaningful comments

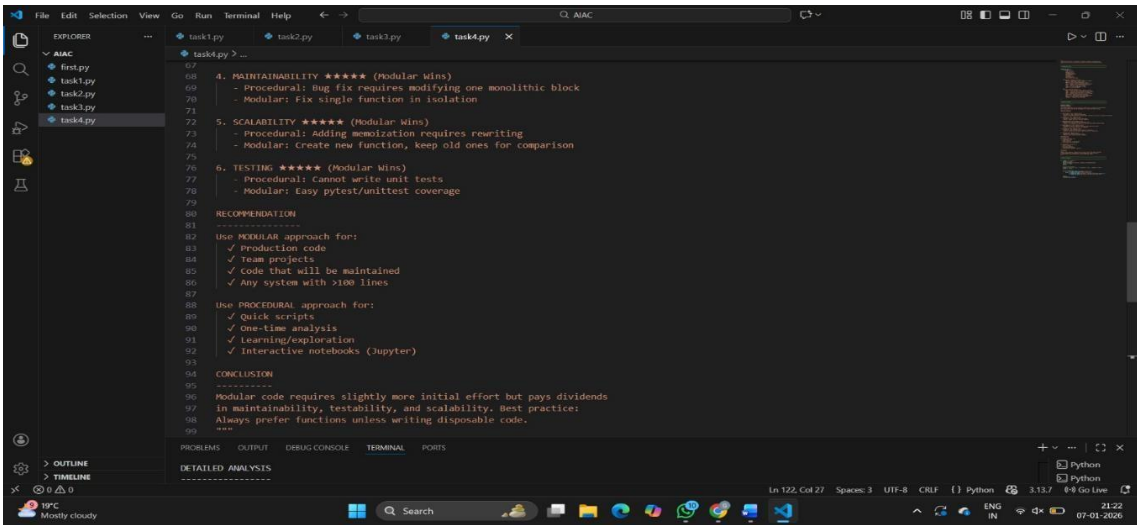


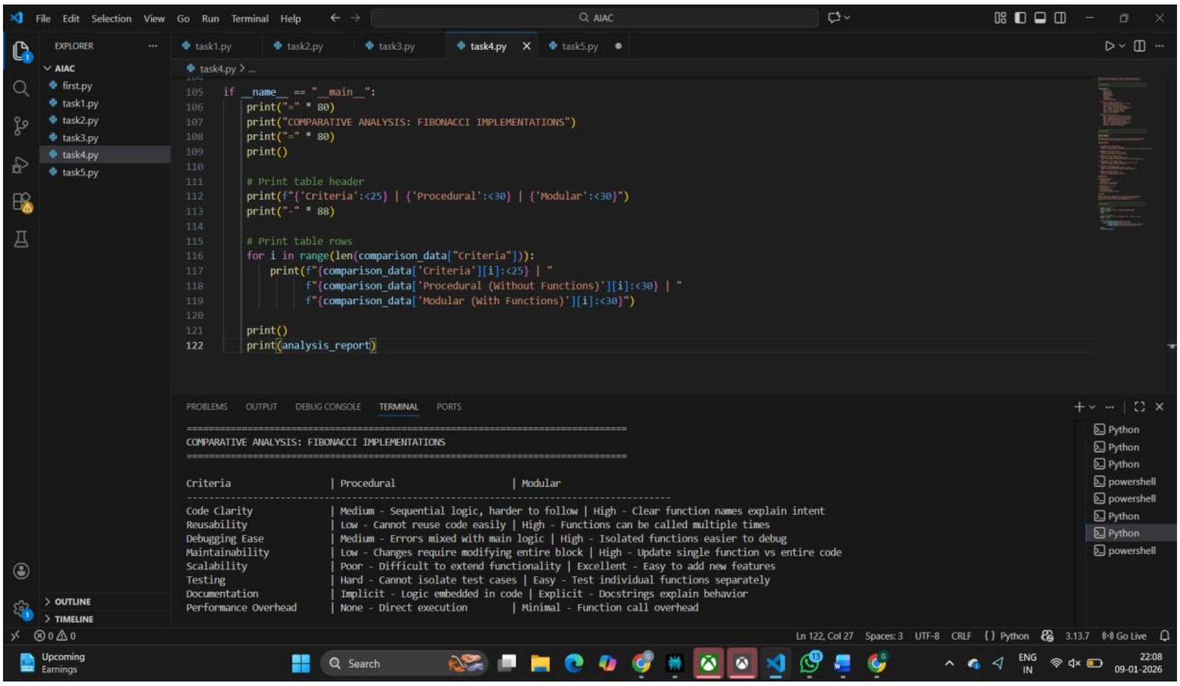
TASK:4

PROMPT: Create a comparative analysis between procedural Fibonacci code without functions and modular Fibonacci code with functions focusing on code clarity, reusability, debugging ease, and suitability for large systems



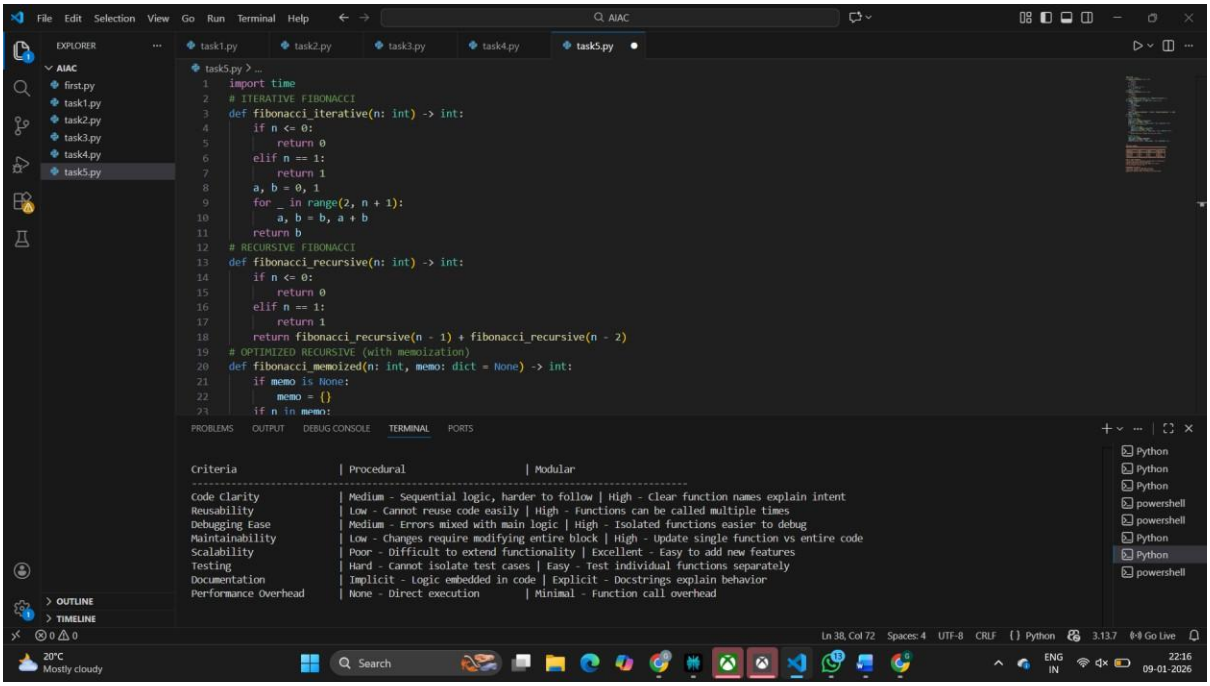


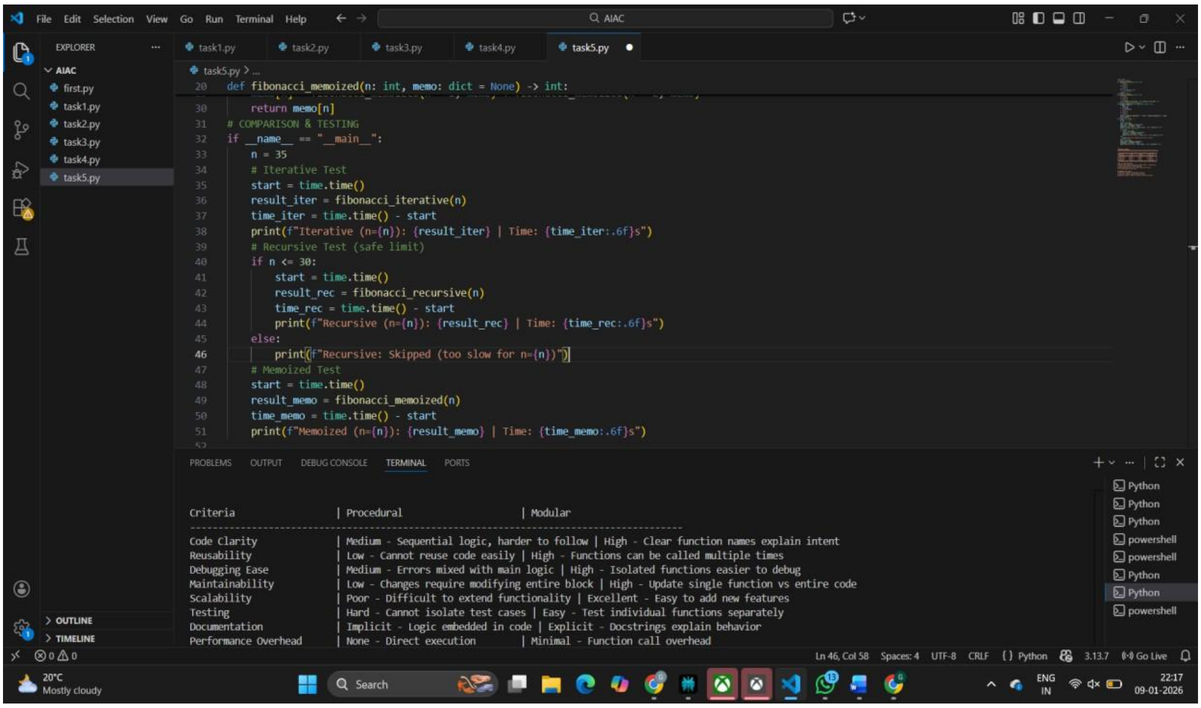




TASK:5

PROMPT: Generate both iterative and recursive Fibonacci implementations in Python with comments, explain their execution flow, and compare their time and space complexity and performance for large n




task5.py
def fibonacci_memoized(n: int, memo: dict = None) -> int:
    return memo[n]
# COMPARISON & TESTING
if __name__ == "__main__":
    n = 35
    # Iterative Test
    start = time.time()
    result_iter = fibonacci_iterative(n)
    time_iter = time.time() - start
    print(f"Iterative (n={n}): {result_iter} | Time: {time_iter:.6f}s")
    # Recursive Test (safe limit)
    if n <= 30:
        start = time.time()
        result_rec = fibonacci_recursive(n)
        time_rec = time.time() - start
        print(f"Recursive (n={n}): {result_rec} | Time: {time_rec:.6f}s")
    else:
        print(f"Recursive: Skipped (too slow for n={n})")
    # Memoized Test
    start = time.time()
    result_memo = fibonacci_memoized(n)
    time_memo = time.time() - start
    print(f"Memoized (n={n}): {result_memo} | Time: {time_memo:.6f}s")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Criteria | Procedural | Modular
Code Clarity | Medium - Sequential logic, harder to follow | High - Clear function names explain intent
Reusability | Low - Cannot reuse code easily | High - Functions can be called multiple times
Debugging Ease | Medium - Errors mixed with main logic | High - Isolated functions easier to debug
Maintainability | Low - Changes require modifying entire block | High - Update single function vs entire code
Scalability | Poor - Difficult to extend functionality | Excellent - Easy to add new features
Testing | Hard - Cannot isolate test cases | Easy - Test individual functions separately
Documentation | Implicit - Logic embedded in code | Explicit - Docstrings explain behavior
Performance Overhead | None - Direct execution | Minimal - Function call overhead
Ln 46, Col 58 Spaces: 4 UTF-8 CRLF Python 3.13.7
20°C Mostly cloudy
22:17 09-01-2026