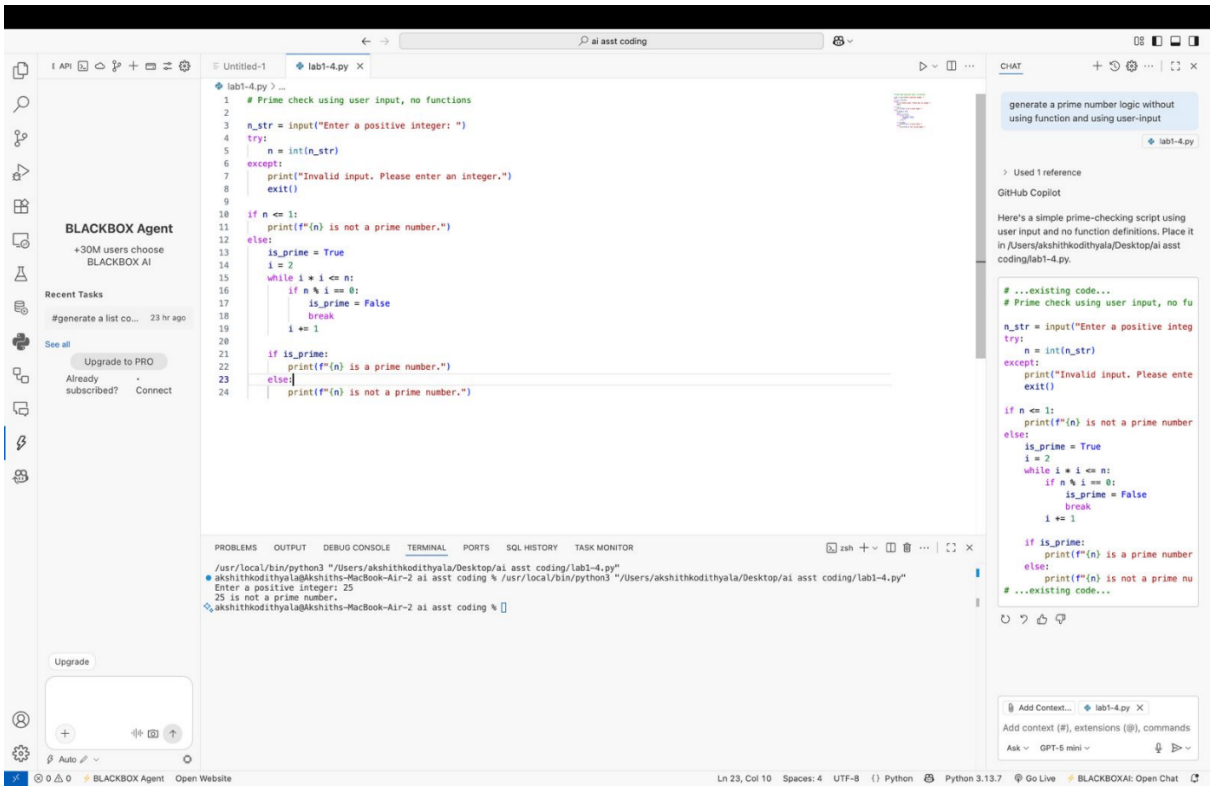
NAME : KODITHYALA AKSHITH

ROLL NO: 2303A52436

BATCH: 44

Task-1



PROMPTS:

Generate a prime number logic without using function and using user-input

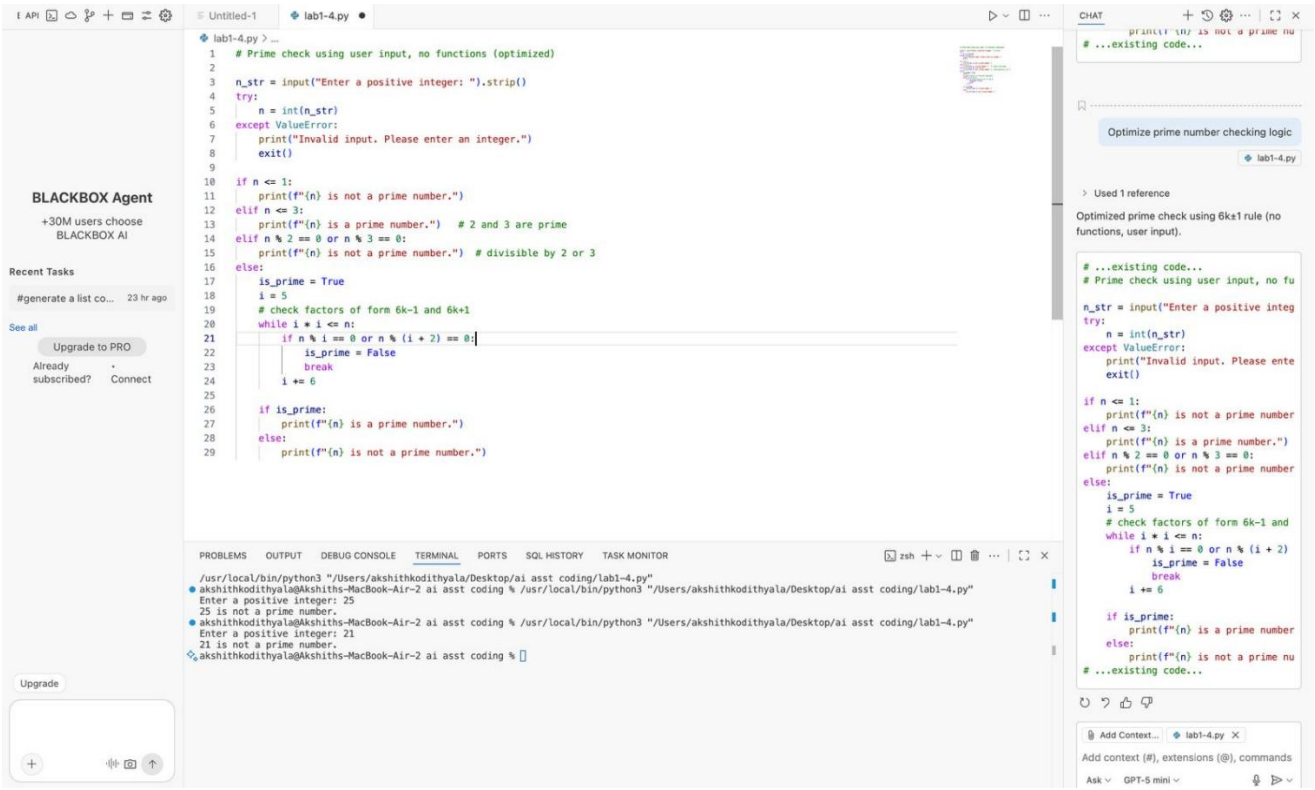
Input: 5

Output: 5 is a prime number

Input:25

Output: 25 is mot a prime number

Task-2



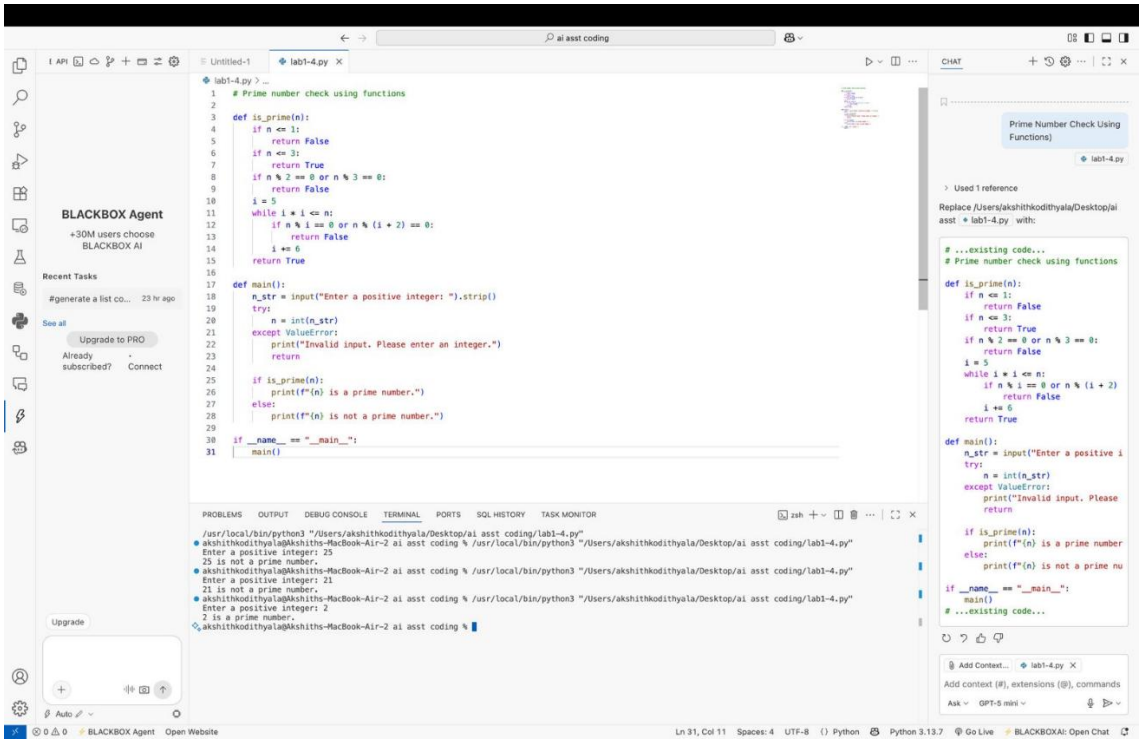
PROMPT:

Optimize prime number checking logic

Original code runs the loop to n times on checking the condition one by one

Since Optimized code runs skipping the odd divisors reducing the number of loops

Task-3



PROMPT:

Prime Number Check Using Functions

Code:

```
def is_prime(num: int) -> bool:
    if num <= 1:
        return False
    if num == 2:
        return True
    if num % 2 == 0:
        return False
    for i in range(3, int(num ** 0.5) + 1, 2):
        if num % i == 0:
            return False
    return True
```

Test Cases:

Input: 5

Output: 5 is a prime number

Input: 2

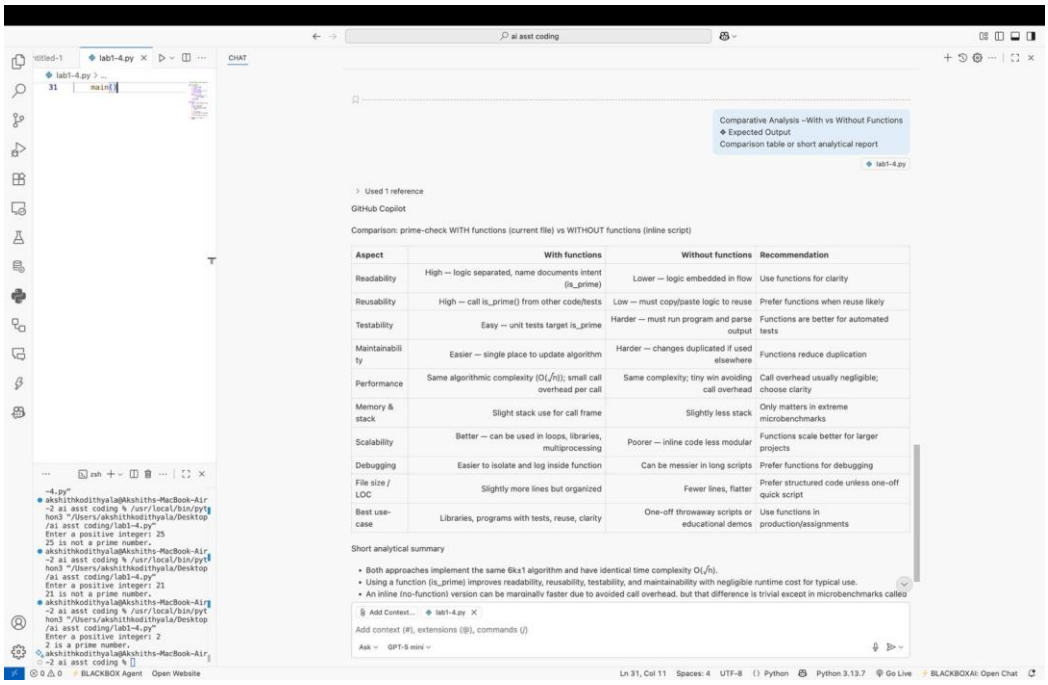
Output: 2 is a prime number

Input: 15

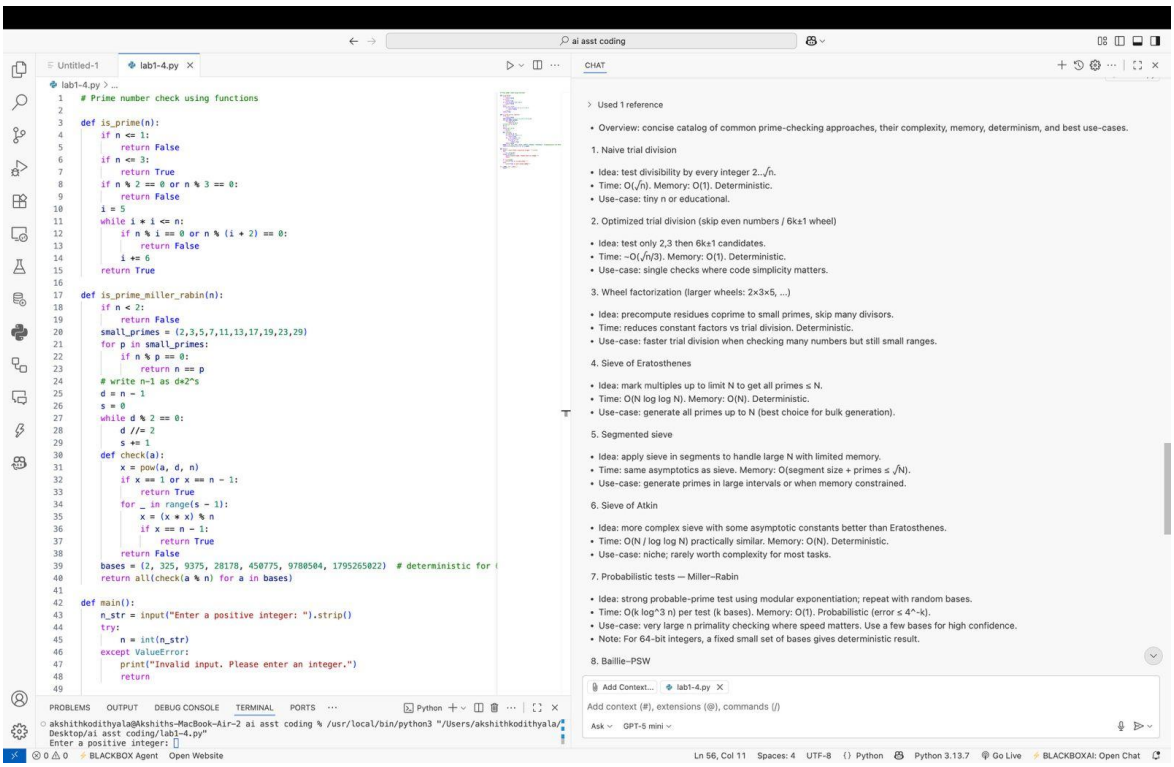
Output: 15 is not a prime number

Task-4

Comparison Table:



TASK-5



┌─ EXECUTION FLOW ──────────────────────────────┐

| WITH FUNCTION: |

| • Main flow → is_prime() call → returns boolean |

| • Function scope isolated from main logic |

| • Single code path, reusable across program |

| |

| WITHOUT FUNCTION: |

| • Inline logic directly in main flow |

| • All conditions evaluated in-place |

| • Must repeat code if used multiple times |

└──┘

┌─ TIME COMPLEXITY ─────────────────────────────┐

| Both approaches: O(√n) |
| • Check divisors from 2 to √n |
| • Skip even numbers (optimization) |
| • Early exit on first divisor found |
| |
| Operations count (approx): |
| • n = 100: ~5 iterations worst case |
| • n = 1,000: ~16 iterations |
| • n = 1,000,000: ~500 iterations |

┌─ PERFORMANCE FOR LARGE INPUTS ─┐

| WITH FUNCTION: |
| ✓ Identical algorithm performance |
| + Function call overhead: ~1-2 microseconds |
| + Negligible for single calls |
| + Better for multiple calls (code reuse) |
| |
| WITHOUT FUNCTION: |
| ✓ Slightly faster (no function call overhead) |
| - Marginal difference: <1% faster |
| - Code duplication increases file size |
| - Harder to optimize if needed later |

┌─ WHEN EACH APPROACH IS APPROPRIATE ─┐

| USE FUNCTION-BASED WHEN: |
| ✓ Checking primality multiple times in code |
| ✓ Building larger programs/libraries |
| ✓ Need unit testing capability |
| ✓ Want clean, maintainable code |

| ✓ Extending logic later (e.g., caching) |

| ✓ Working in teams (better code organization) |

| |

| USE INLINE (NO FUNCTION) WHEN: |

| ✓ One-time check in simple script |

| ✓ Performance-critical (negligible difference) |

| ✓ Learning/prototyping phase |

| ✓ Ultra-minimal code footprint required |

| ⚠ Generally NOT recommended for production |

└──┘