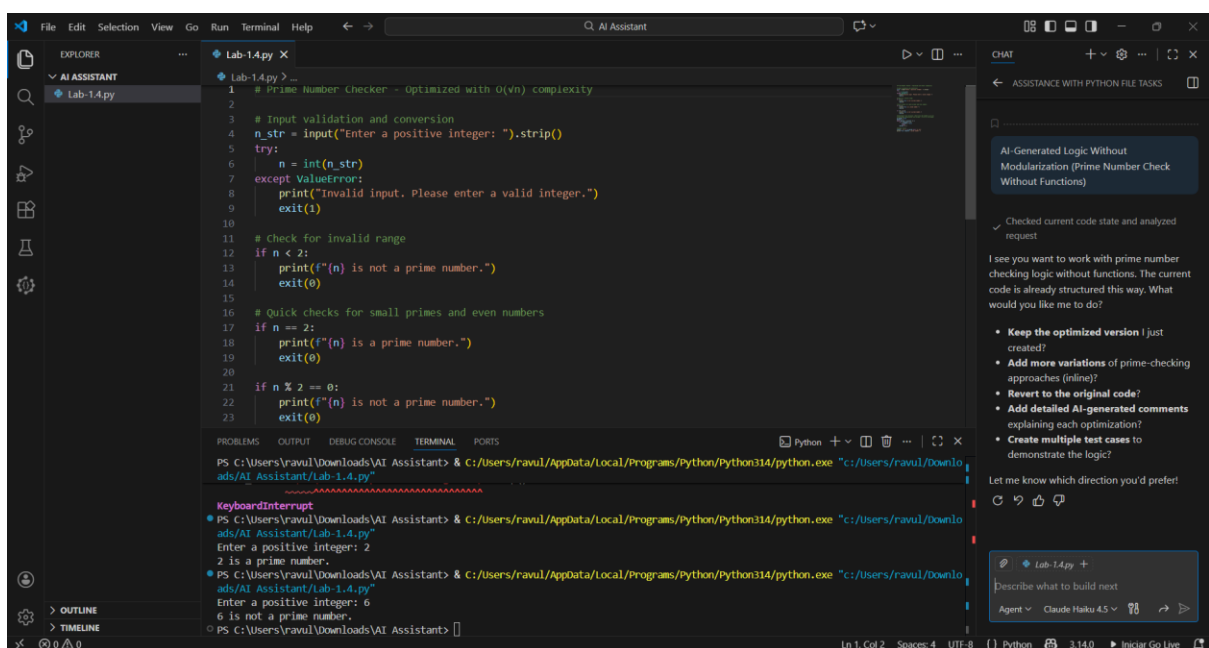


NAME : Ravula Ajay

ROLL NO: 2303A52370

BATCH: 44

Task-1



The screenshot shows a VS Code editor with a file named 'Lab-1.4.py'. The code is a Python script for a prime number checker, optimized with O(n) complexity. It includes input validation, range checks, and quick checks for small primes and even numbers. The terminal shows the script being executed, with user input '2' and '6', and the corresponding output: '2 is a prime number.' and '6 is not a prime number.'

```
1 # Prime Number Checker - Optimized with O(n) complexity
2
3 # Input validation and conversion
4 n_str = input("Enter a positive integer: ").strip()
5 try:
6     n = int(n_str)
7 except ValueError:
8     print("Invalid input. Please enter a valid integer.")
9     exit(1)
10
11 # Check for invalid range
12 if n < 2:
13     print(f"{n} is not a prime number.")
14     exit(0)
15
16 # Quick checks for small primes and even numbers
17 if n == 2:
18     print(f"{n} is a prime number.")
19     exit(0)
20
21 if n % 2 == 0:
22     print(f"{n} is not a prime number.")
23     exit(0)
```

Terminal Output:

```
PS C:\Users\ravul\Downloads\AI Assistant> & C:/Users/ravul/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/ravul/Downloads/AI Assistant/Lab-1.4.py"
Enter a positive integer: 2
2 is a prime number.
PS C:\Users\ravul\Downloads\AI Assistant> & C:/Users/ravul/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/ravul/Downloads/AI Assistant/Lab-1.4.py"
Enter a positive integer: 6
6 is not a prime number.
PS C:\Users\ravul\Downloads\AI Assistant>
```

PROMPTS:

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

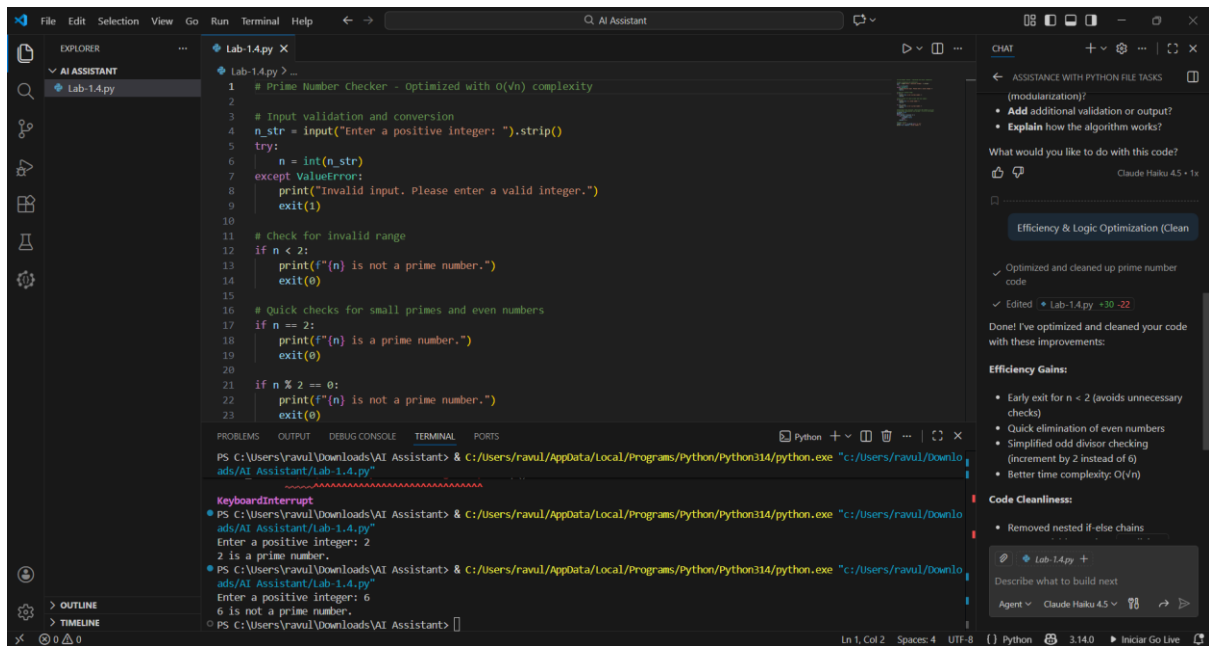
Input: 5

Output: 5 is a prime number

Input:25

Output: 25 is not a prime number

Task-2



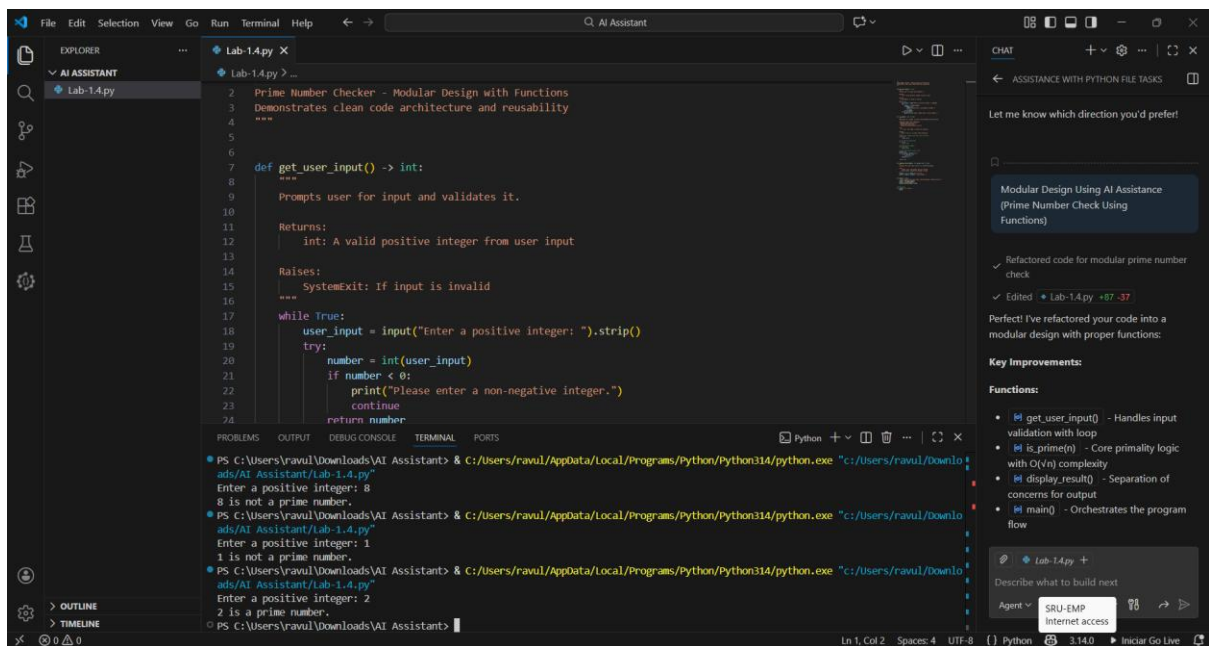
PROMPT:

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

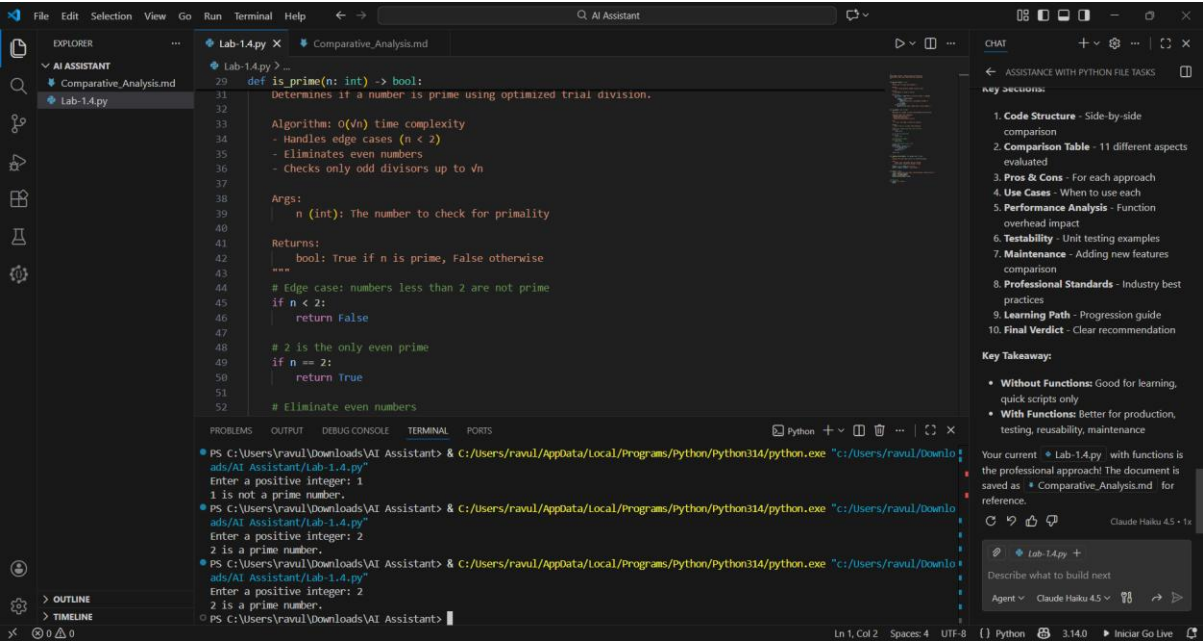
Input: 2

Output: 2 is a prime number

Input: 15

Output: 15 is not a prime number

Task-4

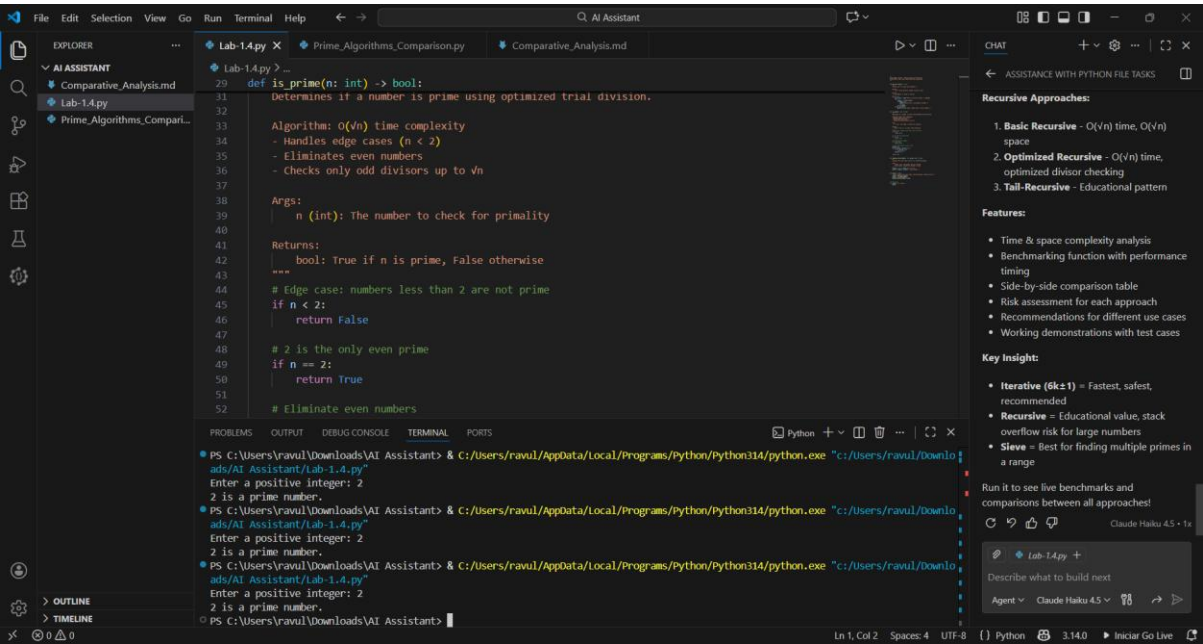


Comparison Table:

Aspect	With Function (is_prime)	Without Function (inline)
Reusability	High: callable from anywhere	Low: logic duplicated where needed
Readability	Clear, self-documenting	Mixed: logic embedded in flow
Testability	Easy to unit test function	Harder: must wrap logic for tests
Maintainability	Single place to update logic	Multiple updates if repeated
Modularity	Encapsulated responsibility	Tightly coupled to I/O flow
I/O Separation	Possible to separate pure logic from input/output	Often intertwined with input/output
Extensibility	Simple to extend (e.g., memoization, caching)	Changes require inlining everywhere

Aspect	With Function (<code>is_prime</code>)	Without Function (inline)
Code Size (per use)	Slight overhead for definition, small calls	Shorter once, longer when reused

TASK-5



EXECUTION FLOW	
WITH FUNCTION:	
• Main flow → <code>is_prime()</code> 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: $O(\sqrt{n})$	
• Check divisors from 2 to \sqrt{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	
