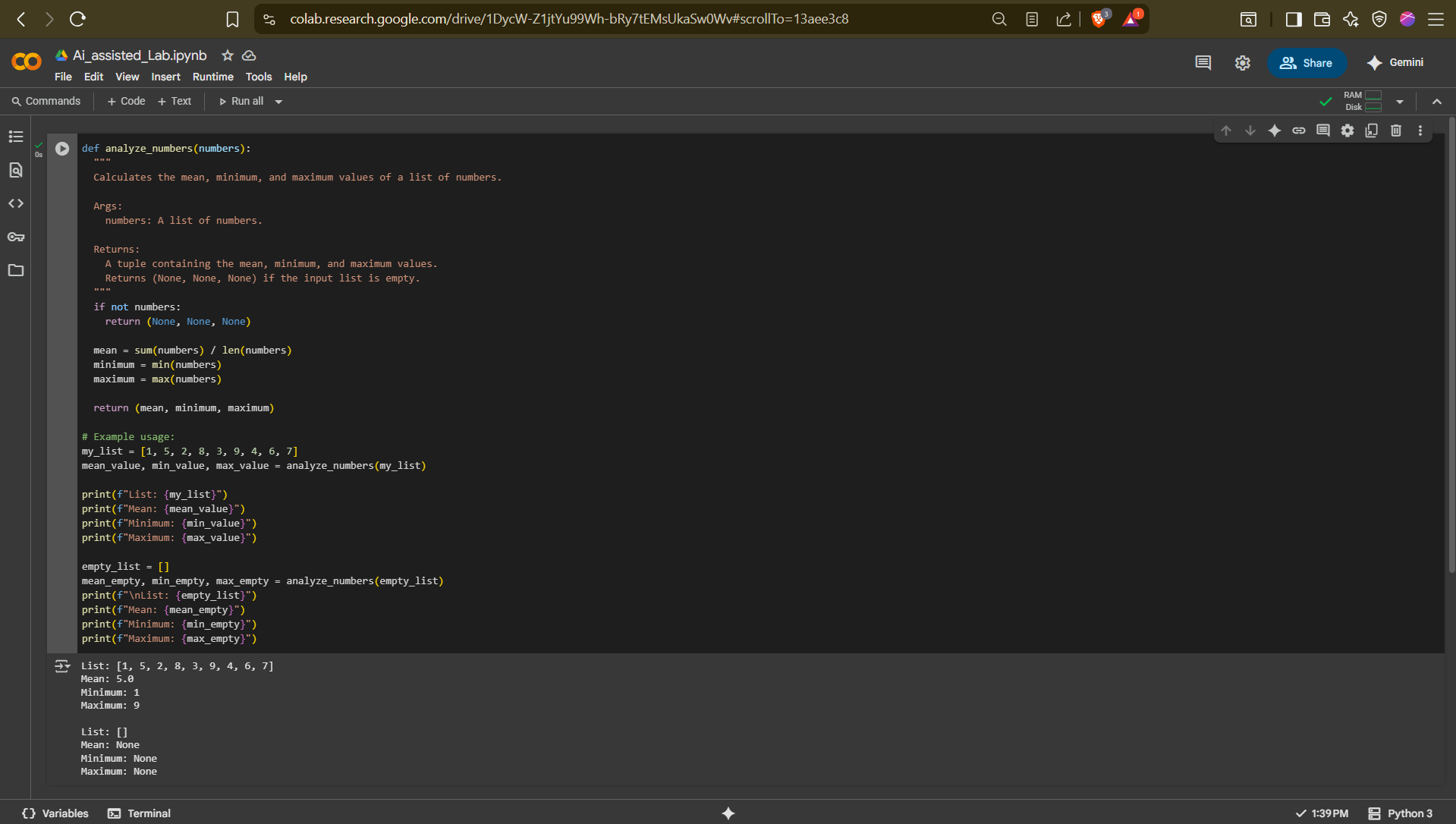
# AI Assisted Coding

**Name:** Nagelli Vishwas

**Roll Number:** 2403A510E4

**Batch:** 24BTCAICSB05

**Task Description #1**



**Task Description #2**

## Comparison Table

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Gemini** | **Copilot** |
| Code Style | Long, detailed, with explanations and edge case checks | Short, direct, no extra checks |
| Comments | Lots of docstrings and notes | Minimal comments |
| Input Handling | Checks for negatives and non-integers | No checks |
| Output | Looks like a tutorial | Ready-to-use snippet |
| Extra Info | No “How it works” section | Adds a short “How it works” |

## Steps, Prompts, and Outputs Documentation

Step 1: Prompting

Prompt used in both AI tools:

Write a Python program to check if a number is an Armstrong number

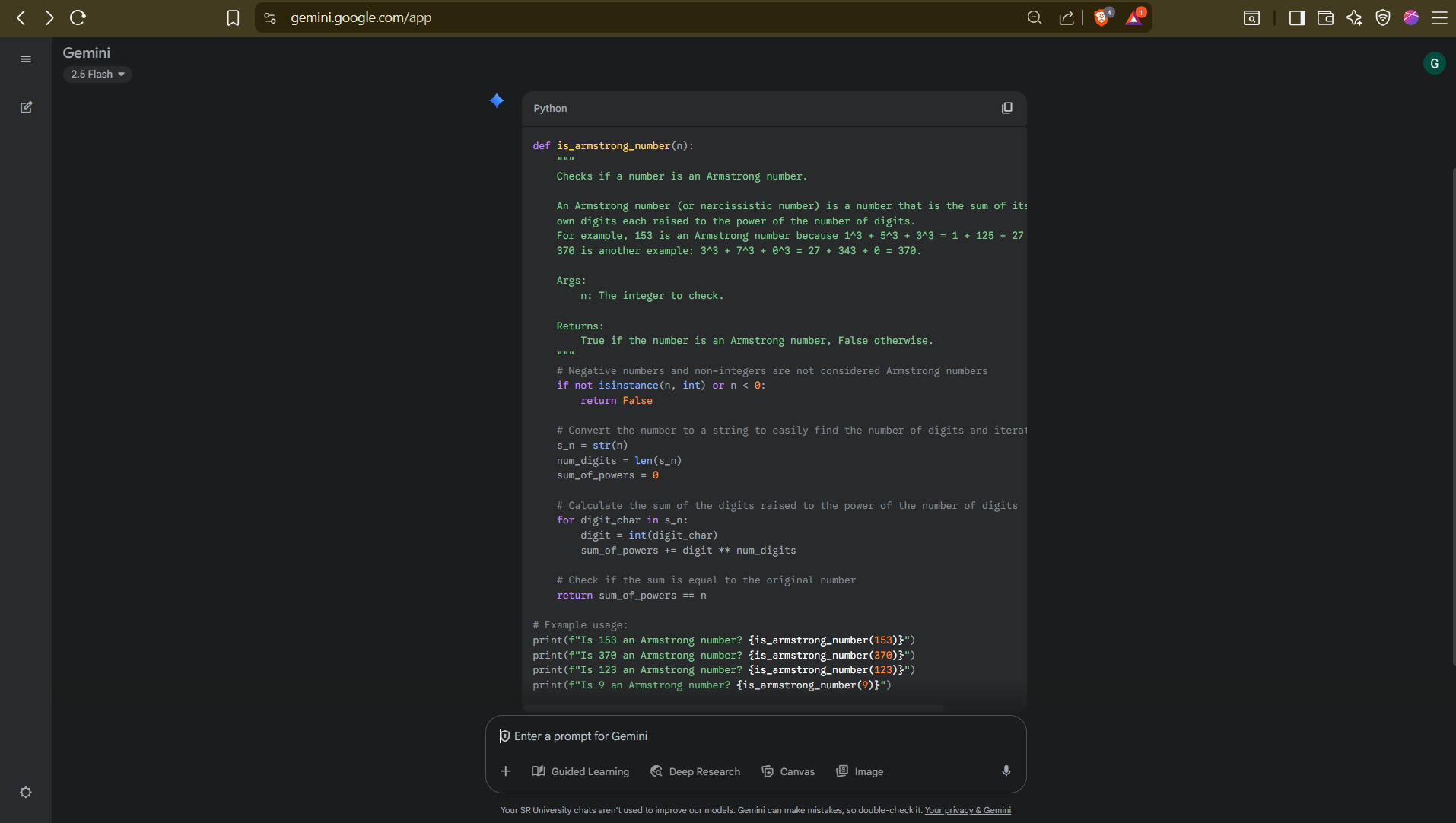
Step 2: Gemini Execution

- Opened Gemini at gemini.google.com/app

- Entered prompt.

- Gemini output: Full docstring, detailed explanation, negative number handling, example with printf style formatting.

- Screenshot:



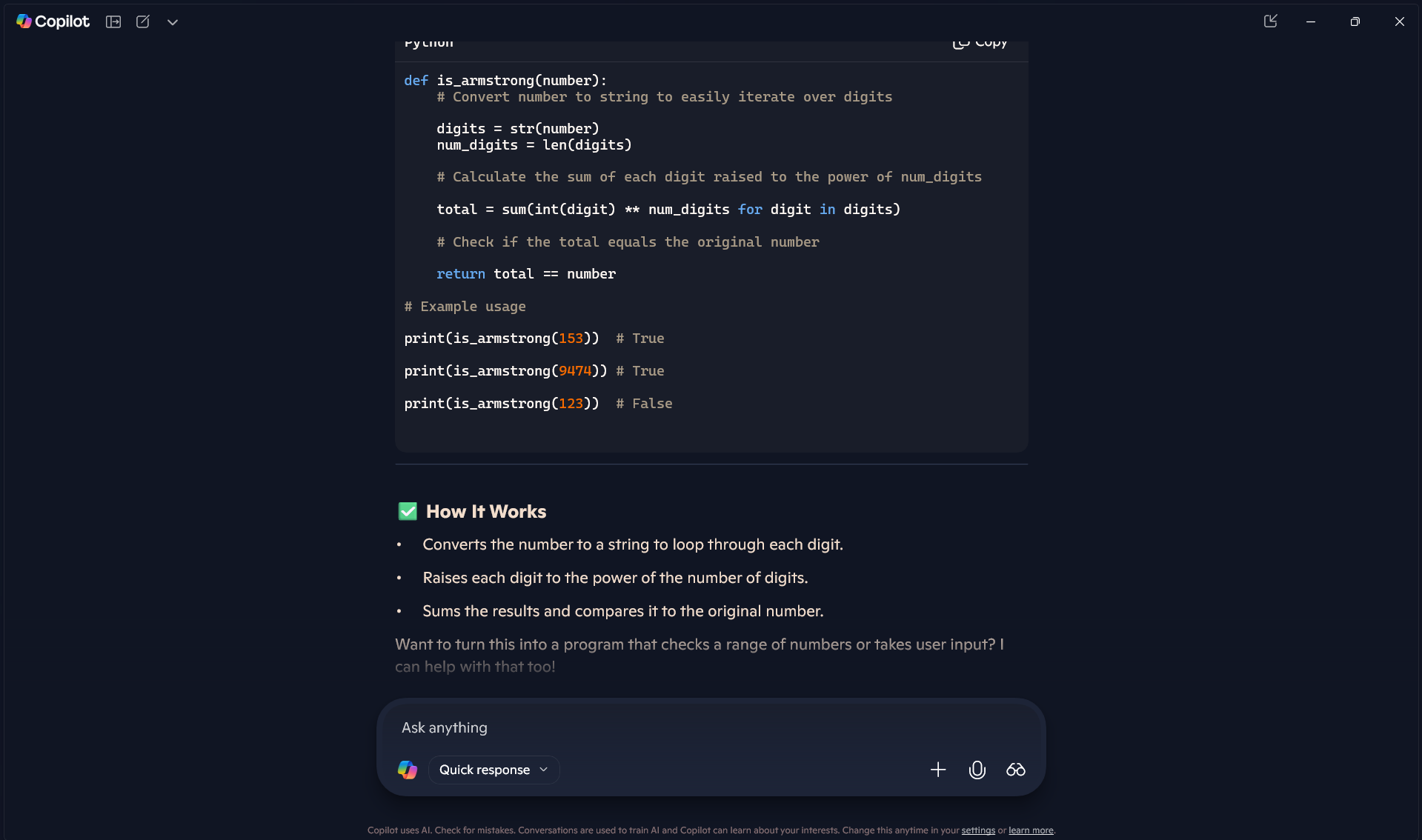
Step 3: Copilot Execution

- Opened GitHub Copilot Chat

- Entered same prompt.

- Copilot output: Minimal code, direct logic, 'How It Works' section, Pythonic print().

- Screenshot:



Step 4: Observations

- Gemini is beginner-friendly with detailed explanations.

- Copilot is concise and better for quick coding tasks.

**Task Description #3**

**Code snippet:**

def is\_palindrome(s):

s = s.lower().replace(" ", "")

return s == s[::-1]

print(is\_palindrome("Racecar"))

print(is\_palindrome("Hello"))

print(is\_palindrome("A man a plan a canal Panama"))

**Gemini’s Explaination:**

Let's break down the is\_palindrome function line by line.

The function determines if a given string is a palindrome, meaning it reads the same forwards and backward.

Python

def is\_palindrome(s):

This line **defines a function** named is\_palindrome that accepts a single argument, a string, which we'll refer to as s within the function.

Python

s = s.lower().replace(" ", "")

This is the core preprocessing step. It modifies the input string s to make the palindrome check more robust by handling different cases and formats.

* s.lower(): This converts the entire string to **lowercase**. For example, "Racecar" becomes "racecar". This makes the check case-insensitive, so "Racecar" and "racecar" are treated as the same.
* .replace(" ", ""): This method removes all spaces from the string. For example, "A man a plan a canal Panama" becomes "amanaplanacanalpanama". The space between the quotation marks "" indicates we're replacing a space character.

The result of these two operations is stored back into the s variable, effectively overwriting the original string.

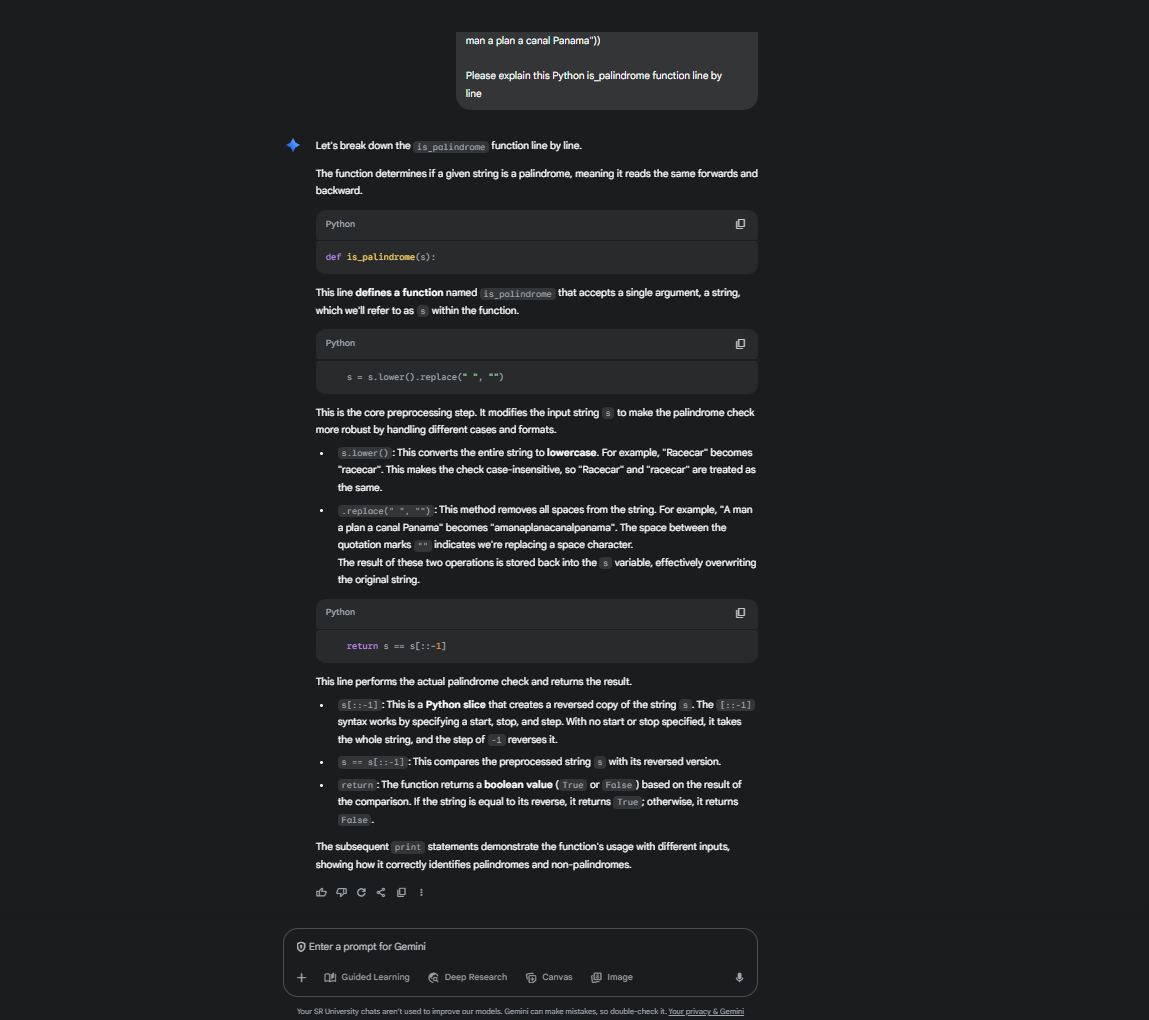
Python

return s == s[::-1]

This line performs the actual palindrome check and returns the result.

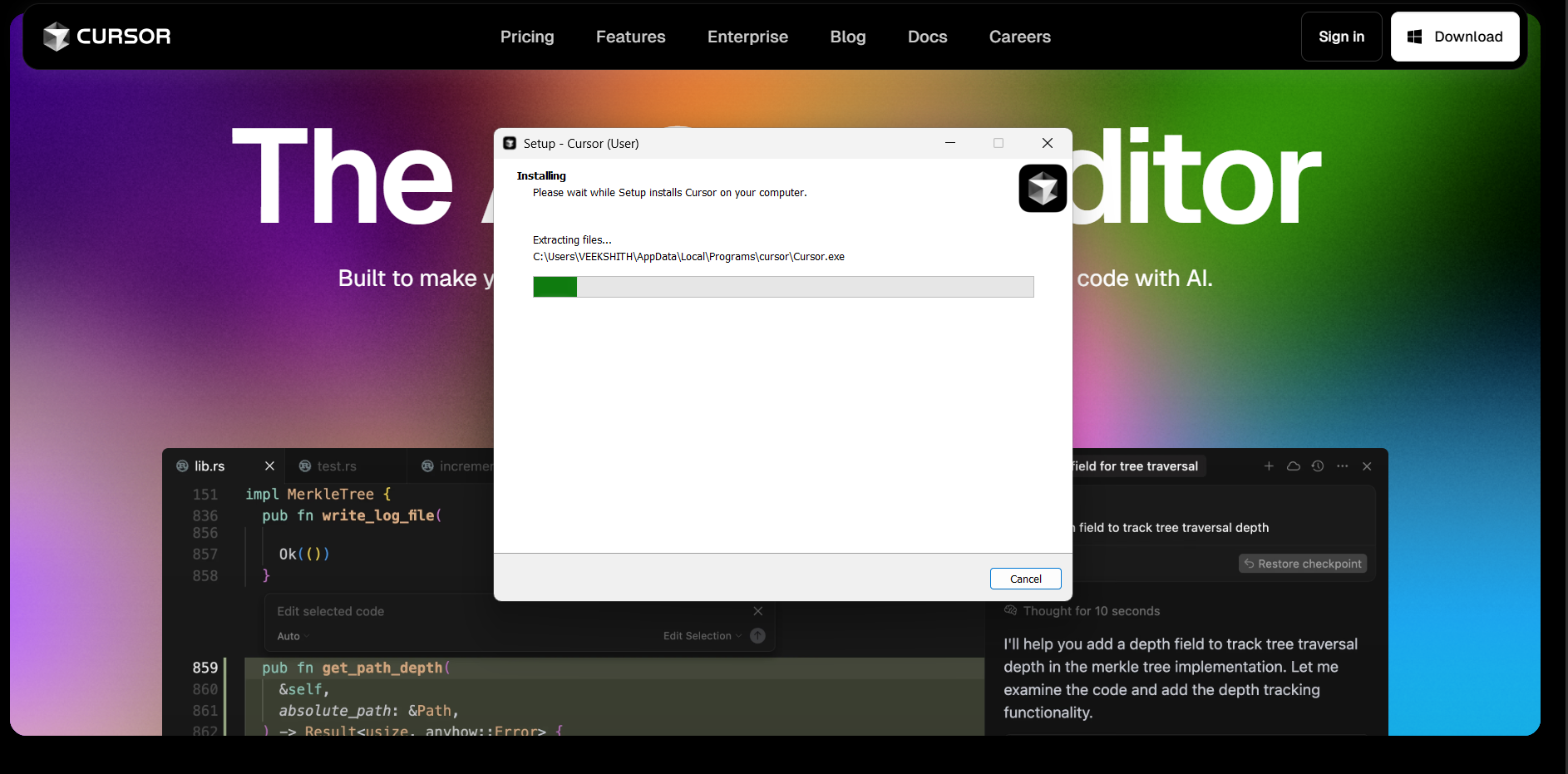
* s[::-1]: This is a **Python slice** that creates a reversed copy of the string s. The [::-1] syntax works by specifying a start, stop, and step. With no start or stop specified, it takes the whole string, and the step of -1 reverses it.
* s == s[::-1]: This compares the preprocessed string s with its reversed version.
* return: The function returns a **boolean value** (True or False) based on the result of the comparison. If the string is equal to its reverse, it returns True; otherwise, it returns False.

The subsequent print statements demonstrate the function's usage with different inputs, showing how it correctly identifies palindromes and non-palindromes.



**Task Description #4**

**Cursor AI setup:**



**Prompts used:**

**Generate a Python function which gives the sum of the first n Natural numbers.**

**Generated code with output:**

def sum\_first\_n(n: int) -> int:

"""

Return the sum of the first n natural numbers (1..n). For n=0, returns 0.

"""

if not isinstance(n, int):

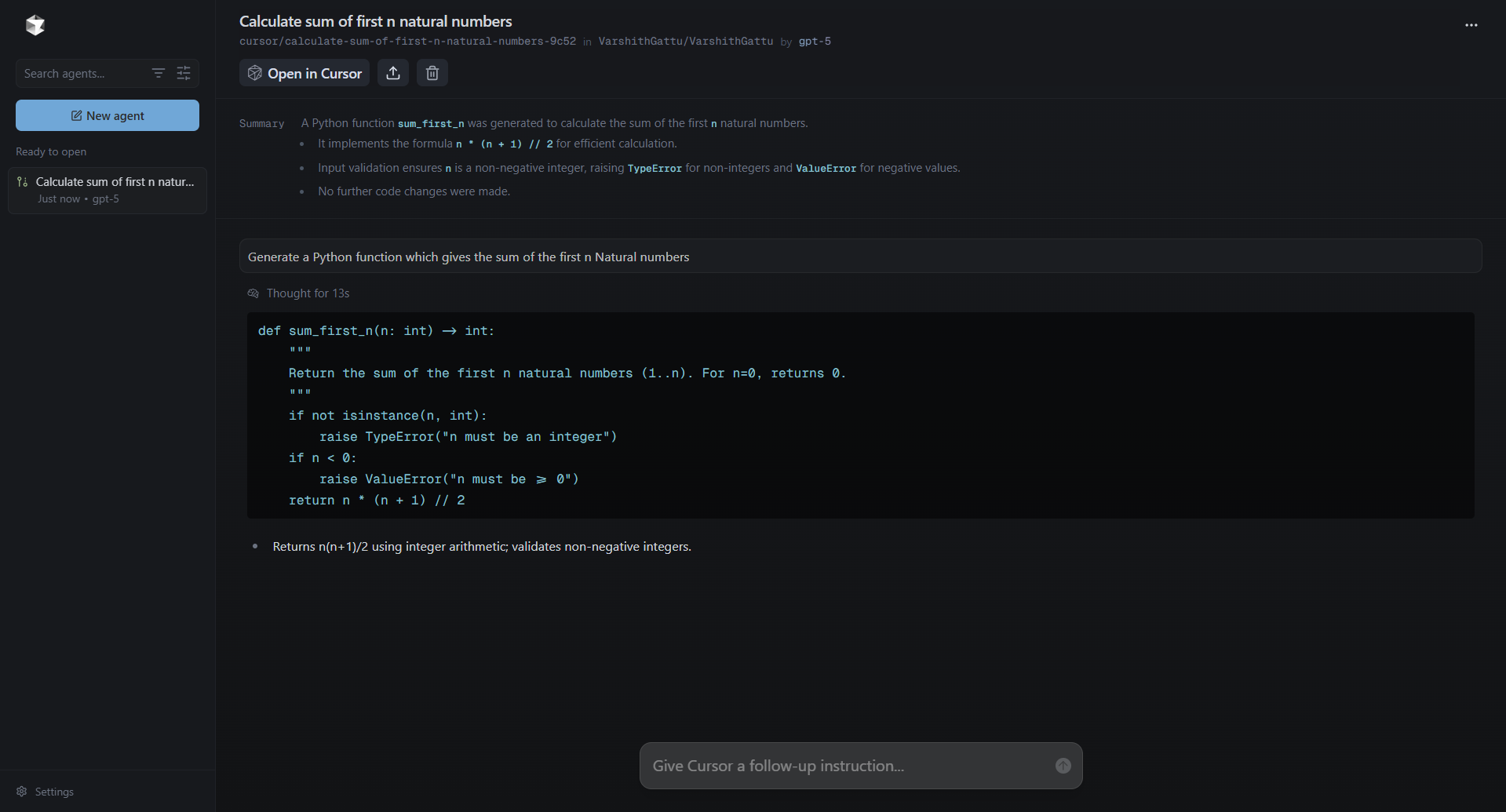
raise TypeError("n must be an integer")

if n < 0:

raise ValueError("n must be >= 0")

return n \* (n + 1) // 2

Returns n(n+1)/2 using integer arithmetic; validates non-negative integers.



**Task Description #5**

**Original basic code:**

numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

even\_sum = 0

odd\_sum = 0

for num in numbers:

if num % 2 == 0:

even\_sum += num

else:

odd\_sum += num

print("Sum of even numbers:", even\_sum)

print("Sum of odd numbers:", odd\_sum)

**Sample Output:**

Sum of even numbers: 30

Sum of odd numbers: 25

**Refactored code:**

def sum\_even\_odd(nums):

even\_sum = sum(n for n in nums if n % 2 == 0)

odd\_sum = sum(n for n in nums if n % 2 != 0)

return even\_sum, odd\_sum

numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

even\_total, odd\_total = sum\_even\_odd(numbers)

print(f"Sum of even numbers: {even\_total}")

print(f"Sum of odd numbers: {odd\_total}")

**Sample Output:**

Sum of even numbers: 30

Sum of odd numbers: 25

**Explanation of refactored code:**

**Line 1:** Comment describing what the code does — calculates the sum of odd and even numbers in a tuple.

**Line 3:** Defines a function sum\_even\_odd with one parameter nums, which is expected to be a tuple of integers.

**Line 4:** A **docstring** explaining that the function returns the sum of even and odd numbers from the given tuple.

**Line 5:** Uses the sum() function with a generator expression to add up all numbers n in nums that are divisible by 2 (n % 2 == 0), meaning they are even.

**Line 6:** Similarly, sums up all numbers n in nums that are not divisible by 2 (n % 2 != 0), meaning they are odd.

**Line 7:** Returns both even\_sum and odd\_sum as a tuple.

**Line 10:** Creates a sample tuple numbers containing integers from 1 to 10.

**Line 13:** Calls sum\_even\_odd(numbers) and stores the two returned values in even\_total and odd\_total.

**Line 16:** Prints the sum of even numbers using an f-string for clear formatting.

**Line 17:** Prints the sum of odd numbers using an f-string.

**Output Screenshot:**

