

AI Assistant Coding

Assignment-2.5

Name: **S Srinidhi**

HT.NO.: **2303A51342**

Batch:**20**

Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI

Task 1: Refactoring Odd/Even Logic (List Version)

Scenario

You are improving legacy code that calculates the sum of odd and even numbers in a list.

Prompt Used (Gemini / Cursor AI):

"Write a Python program to calculate the sum of odd and even numbers in a list, then refactor it to improve readability and efficiency."

Original Code (Legacy Version)

```
numbers = [10, 15, 22, 33, 40]

odd_sum = 0

even_sum = 0

for i in range(len(numbers)):

    if numbers[i] % 2 == 0:

        even_sum = even_sum + numbers[i]

    else:

        odd_sum = odd_sum + numbers[i]
```

```
print("Odd Sum:", odd_sum)

print("Even Sum:", even_sum)
```

Refactored Code (AI-Improved Version)

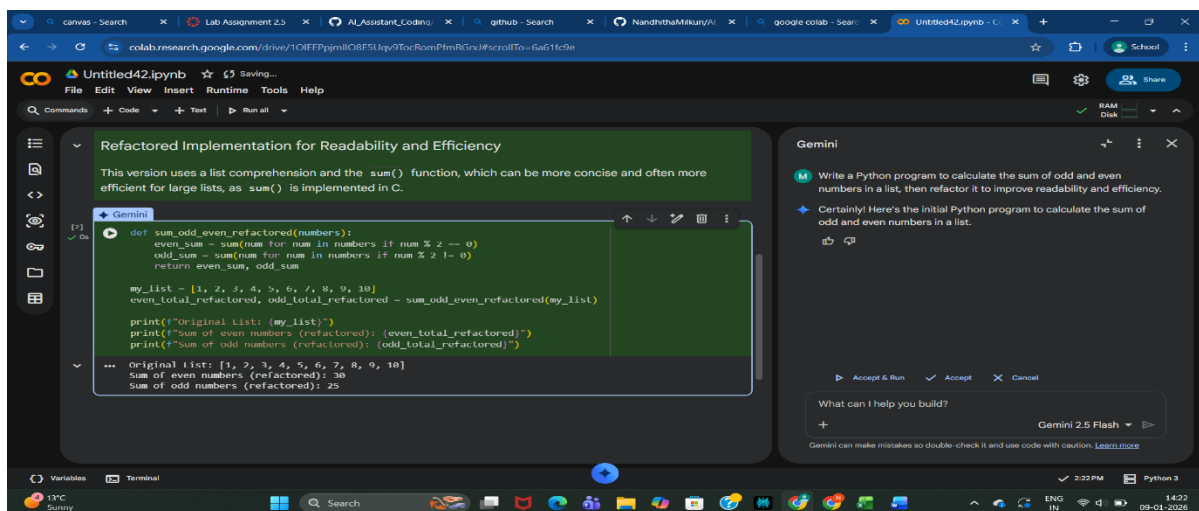
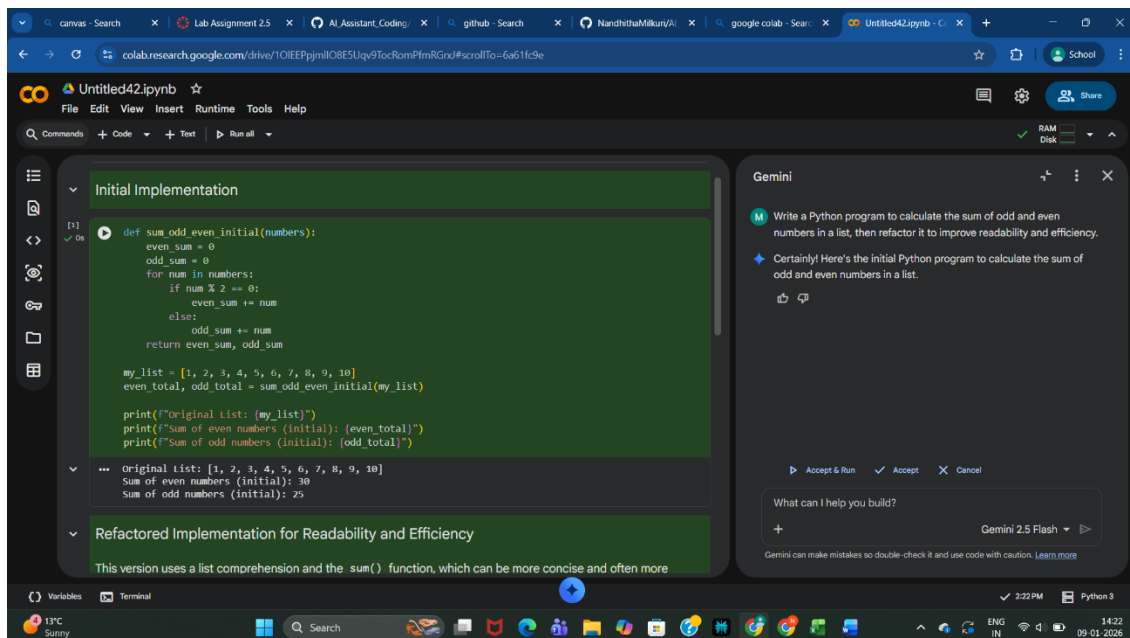
```
numbers = [10, 15, 22, 33, 40]

odd_sum = sum(num for num in numbers if num % 2 != 0)

even_sum = sum(num for num in numbers if num % 2 == 0)

print(f"Odd Sum: {odd_sum}")

print(f"Even Sum: {even_sum}")
```



Explanation:

The refactored version improves readability by using Python's built-in `sum()` function and list comprehensions. It reduces the number of lines and avoids manual looping, making the code cleaner and more efficient.

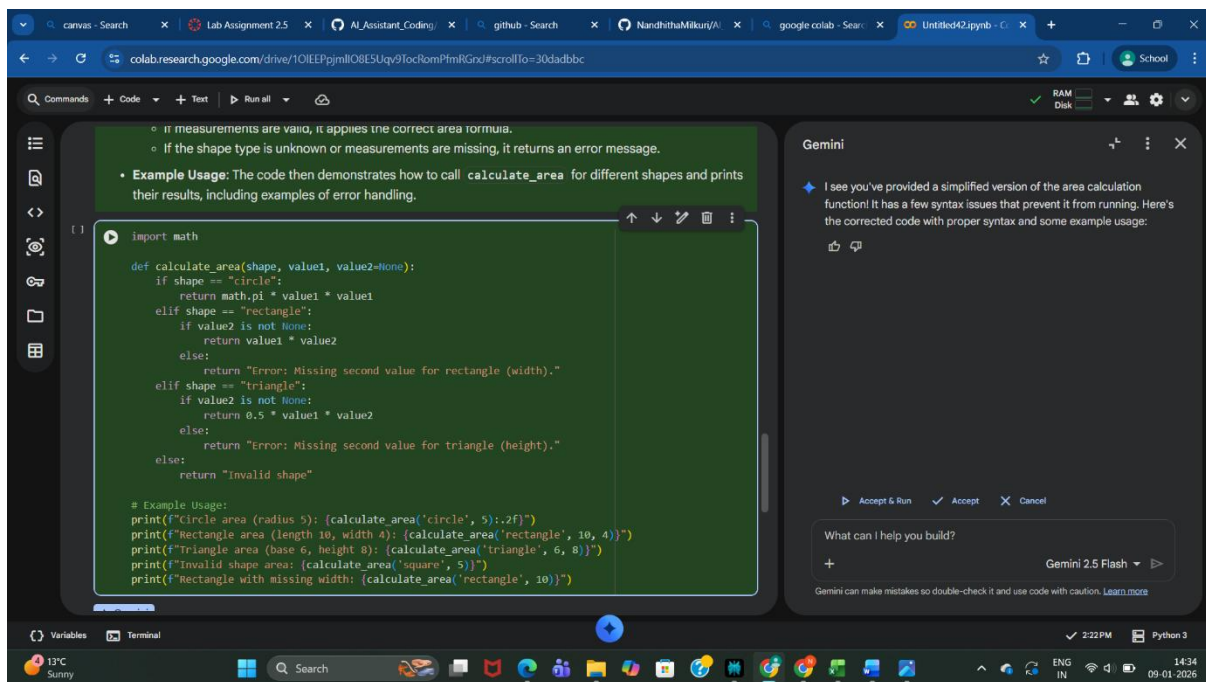
Task 2: Area Calculation Explanation

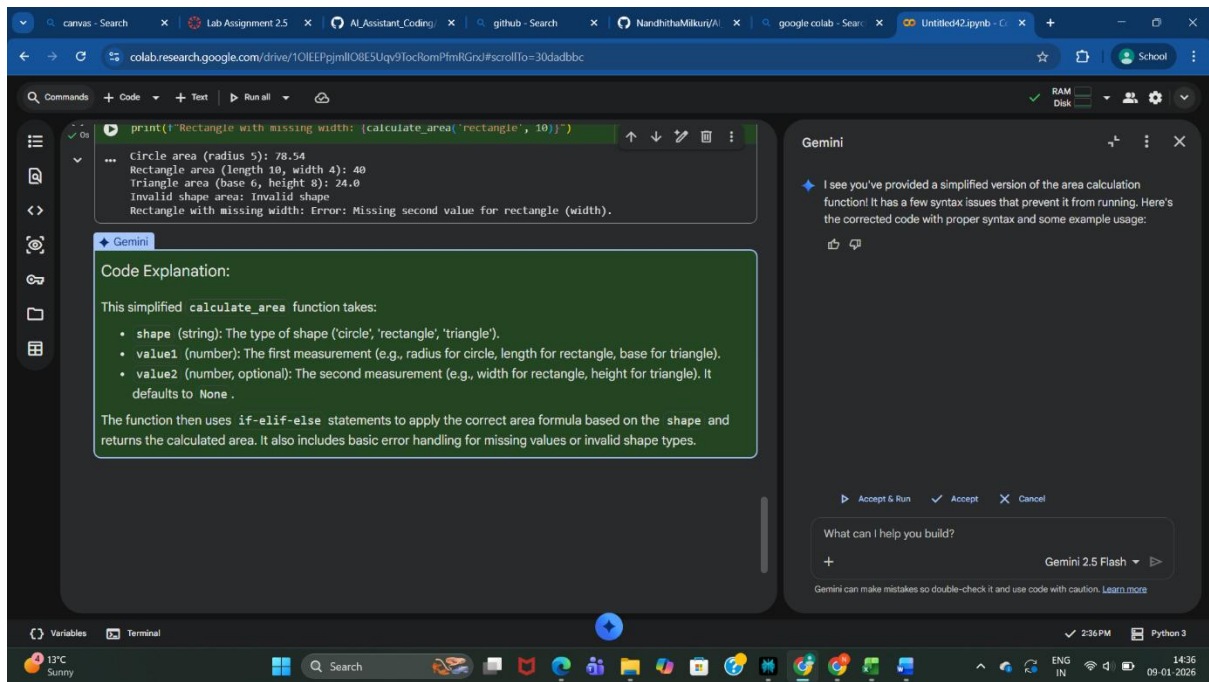
Scenario

You are onboarding a junior developer and want a clear explanation of a function that calculates the area of different shapes.

Prompt Used (Google Gemini):

"Explain a Python function that calculates the area of different shapes like circle, rectangle, and triangle in simple terms."





Code

```
import math

def calculate_area(shape, value1, value2=None):

    if shape == "circle":

        return math.pi * value1 * value1

    elif shape == "rectangle":

        return value1 * value2

    elif shape == "triangle":

        return 0.5 * value1 * value2

    else:

        return "Invalid shape"
```

Explanation

This function calculates the area based on the shape name passed as input. For a circle, it uses the formula πr^2 where value1 is the radius. For a rectangle, it multiplies length and breadth. For a triangle, it applies the formula $\frac{1}{2} \times \text{base} \times \text{height}$. The function checks the shape type using conditional statements and returns the appropriate area.

Task 3: Prompt Sensitivity Experiment

Scenario

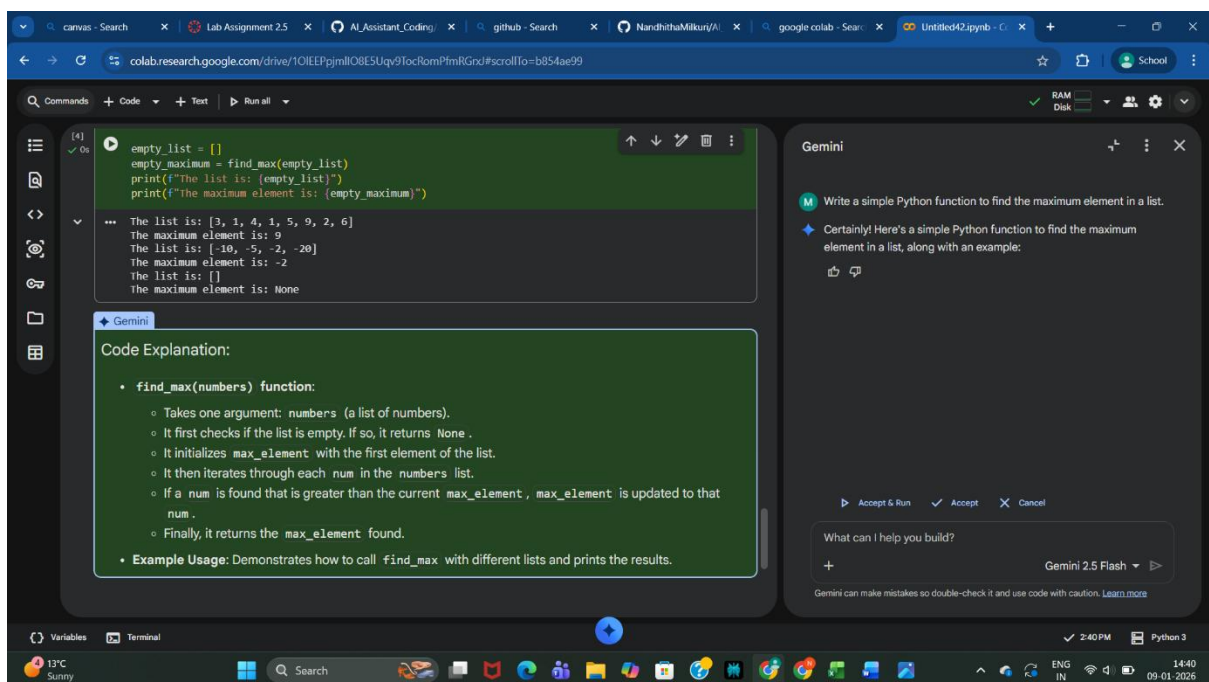
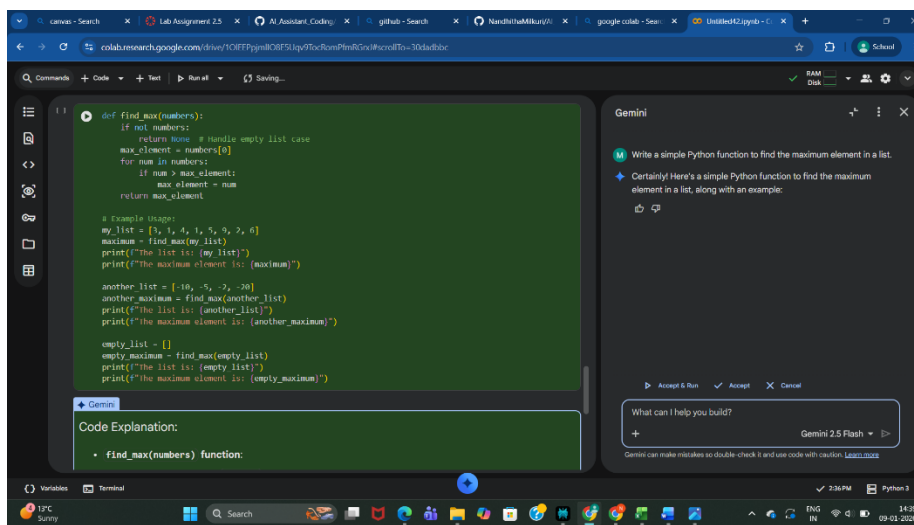
You are testing how Cursor AI responds to different prompts for the same problem.

Problem Statement

"Write a Python function to find the maximum number in a list."

Prompt 1

"Write a simple Python function to find the maximum element in a list."

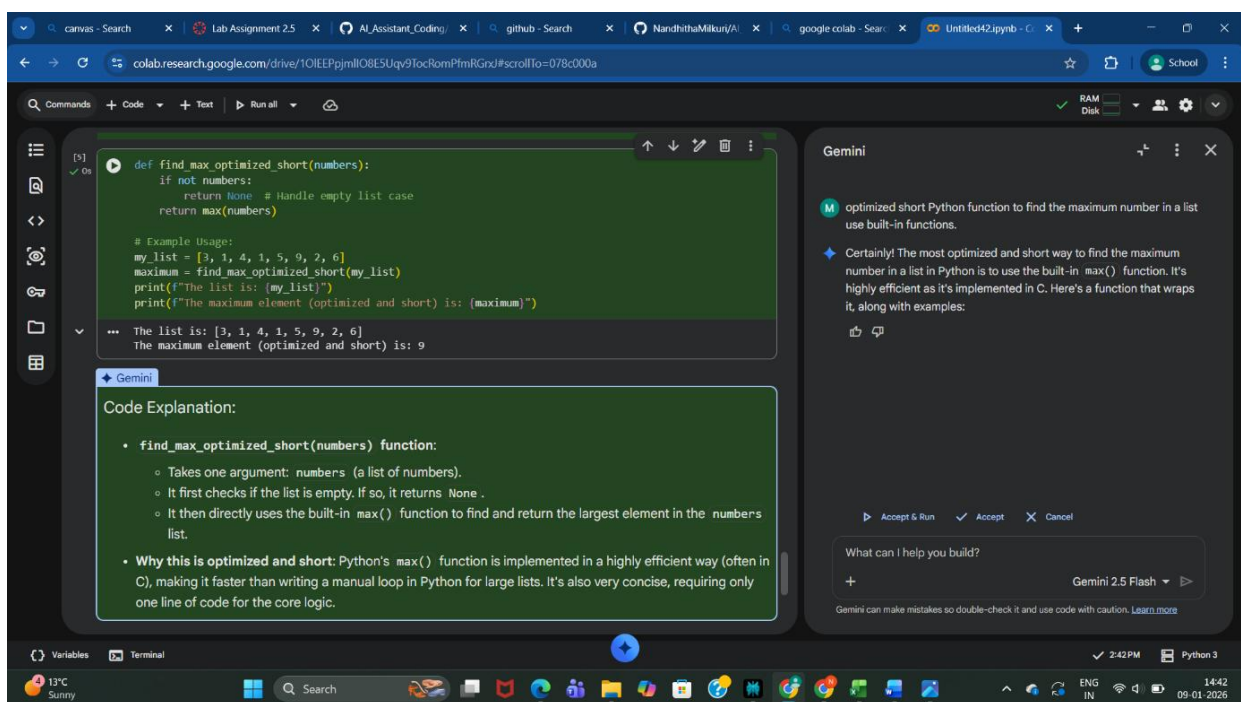


Code:

```
def find_max(nums):  
    max_value = nums[0]  
    for num in nums:  
        if num > max_value:  
            max_value = num  
    return max_value
```

Prompt 2

"Write an optimized Python function to find the maximum number in a list using built-in functions."

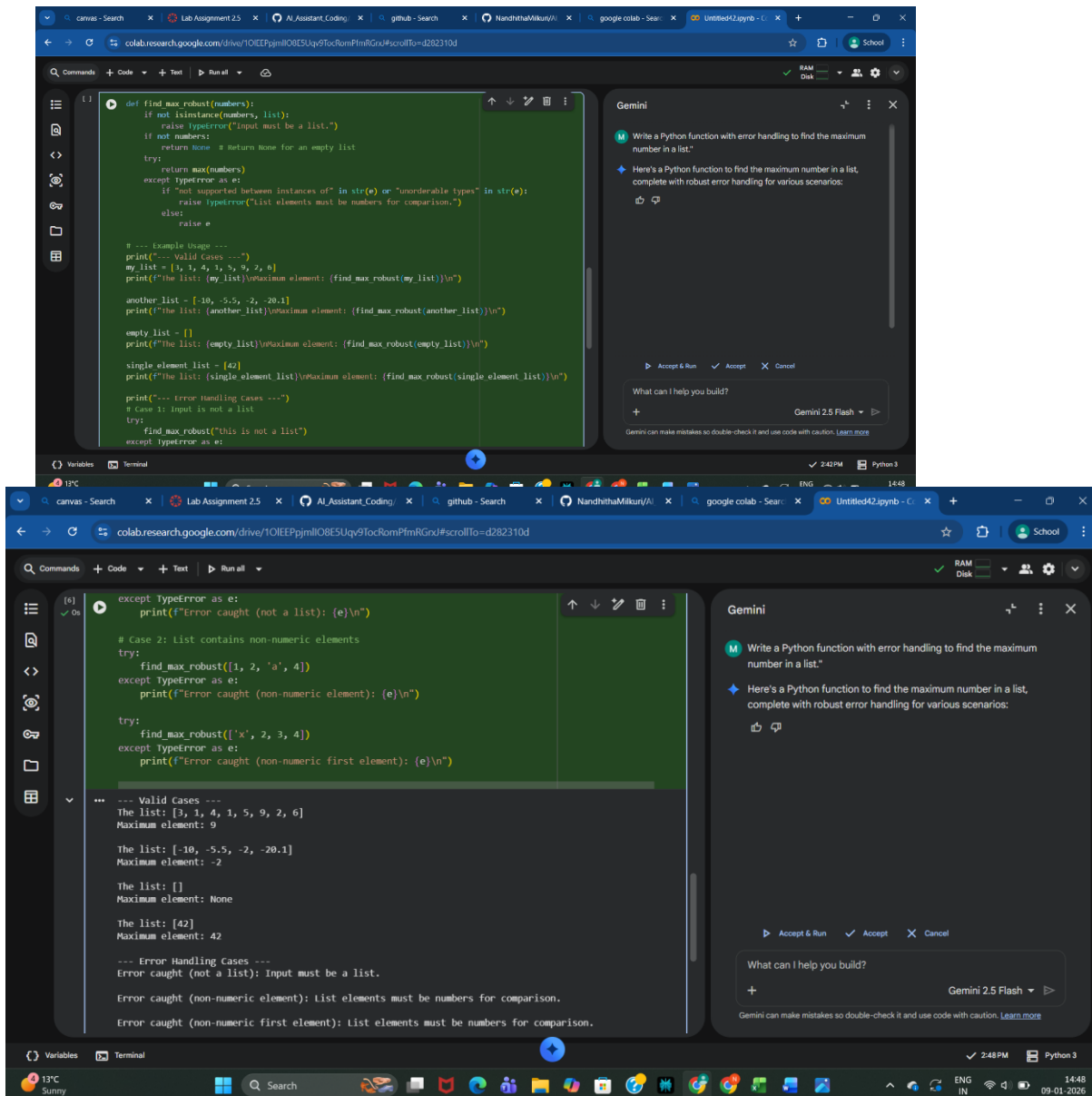


Output Code

```
def find_max(nums):  
    return max(nums)
```

Prompt 3

"Write a Python function with error handling to find the maximum number in a list."



```
def find_max_robust(numbers):
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list.")
    if not numbers:
        return None # Return None for an empty list
    try:
        return max(numbers)
    except TypeError as e:
        if "not supported between instances of" in str(e) or "unorderable types" in str(e):
            raise TypeError("List elements must be numbers for comparison.")
        else:
            raise e

# --- Example Usage ---
print("--- Valid Cases ---")
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
print(f"The list: {my_list} \ Maximum element: {find_max_robust(my_list)} \n")

another_list = [-10, -5.5, -2, -20.1]
print(f"The list: {another_list} \ Maximum element: {find_max_robust(another_list)} \n")

empty_list = []
print(f"The list: {empty_list} \ Maximum element: {find_max_robust(empty_list)} \n")

single_element_list = [42]
print(f"The list: {single_element_list} \ Maximum element: {find_max_robust(single_element_list)} \n")

print("--- Error Handling Cases ---")
# Case 1: Input is not a list
try:
    find_max_robust("this is not a list")
except TypeError as e:
    print(e)
```

What can I help you build?

Gemini 2.5 Flash

```
except TypeError as e:
    print(f"Error caught (not a list): {e} \n")

# Case 2: List contains non-numeric elements
try:
    find_max_robust([1, 2, 'a', 4])
except TypeError as e:
    print(f"Error caught (non-numeric element): {e} \n")

try:
    find_max_robust(['x', 2, 3, 4])
except TypeError as e:
    print(f"Error caught (non-numeric first element): {e} \n")

--- Valid Cases ---
The list: [3, 1, 4, 1, 5, 9, 2, 6]
Maximum element: 9

The list: [-10, -5.5, -2, -20.1]
Maximum element: -2

The list: []
Maximum element: None

The list: [42]
Maximum element: 42

--- Error Handling Cases ---
Error caught (not a list): Input must be a list.

Error caught (non-numeric element): List elements must be numbers for comparison.

Error caught (non-numeric first element): List elements must be numbers for comparison.
```

What can I help you build?

Gemini 2.5 Flash

Output Code

```
def find_max_robust(numbers):

    if not isinstance(numbers, list):

        raise TypeError("Input must be a list.")

    if not numbers:

        return None # Return None for an empty list

    try:
```

```

    return max(numbers)

except TypeError as e:
    if "not supported between instances of" in str(e) or "unorderable types" in str(e):
        raise TypeError("List elements must be numbers for comparison.")
    else:
        raise e

# --- Example Usage ---

print("--- Valid Cases ---")

my_list = [3, 1, 4, 1, 5, 9, 2, 6]

print(f"The list: {my_list}\nMaximum element: {find_max_robust(my_list)}\n")

another_list = [-10, -5.5, -2, -20.1]

print(f"The list: {another_list}\nMaximum element: {find_max_robust(another_list)}\n")

empty_list = []

print(f"The list: {empty_list}\nMaximum element: {find_max_robust(empty_list)}\n")

single_element_list = [42]

print(f"The list: {single_element_list}\nMaximum element: {find_max_robust(single_element_list)}\n")

print("--- Error Handling Cases ---")

# Case 1: Input is not a list

try:
    find_max_robust("this is not a list")
except TypeError as e:
    print(f"Error caught (not a list): {e}\n")

# Case 2: List contains non-numeric elements

try:

```



```

    find_max_robust([1, 2, 'a', 4])
except TypeError as e:
    print(f"Error caught (non-numeric element): {e}\n")
try:
    find_max_robust(['x', 2, 3, 4])
except TypeError as e:
    print(f"Error caught (non-numeric first element): {e}\n")

```

Observation

Cursor AI clearly demonstrates prompt sensitivity, meaning the quality, structure, and safety of the generated code change based on how the prompt is written.

- In Prompt 1, the instruction is very basic, so Cursor AI produces a traditional loop-based solution. This version is easy to understand for beginners and shows the step-by-step logic of finding a maximum value manually.
- In Prompt 2, the word *optimized* guides Cursor AI to use Python's built-in `max()` function. This results in shorter, cleaner, and more efficient code with better readability and performance.
- In Prompt 3, the phrase *with error handling* makes Cursor AI consider edge cases. The generated code checks whether the list is empty before applying `max()`, preventing runtime errors and improving reliability.

Task 4: Tool Comparison Reflection

Reflection

Google Gemini is best for learning and understanding code because it explains concepts clearly in simple language. It is very helpful for students and beginners who want to understand logic step by step. GitHub Copilot is useful for fast coding and productivity, especially for experienced developers working on real projects. However, it focuses more on code completion than explanation. Cursor AI is strong in refactoring and experimenting with different prompts, helping improve code quality, readability, and error handling. Each tool serves a different purpose based on learning, speed, or code improvement needs, making them suitable for different stages of software development.