

AI Assignment2:

Task 1:

Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values

The image consists of two screenshots of a Google Colab notebook interface, showing the process of writing and testing a Python function.

Top Screenshot: The notebook is titled "AL2.pynb". The code editor shows a function definition:

```
def calculate_stats(numbers):  
    """  
    calculates the mean, minimum, and maximum values of a list of numbers.  
  
    Args:  
        numbers: A list of numbers.  
  
    Returns:  
        A dictionary containing the mean, minimum, and maximum values.  
        Returns None if the input list is empty.  
    """  
    if not numbers:  
        return None  
  
    mean_value = sum(numbers) / len(numbers)  
    min_value = min(numbers)  
    max_value = max(numbers)  
  
    return {  
        'mean': mean_value,  
        'minimum': min_value,  
        'maximum': max_value  
    }  
  
# Example usage  
my_list = [10, 20, 30, 40, 50]  
stats = calculate_stats(my_list)  
  
if stats:
```

The right sidebar shows the Gemini chat interface with the prompt: "write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values." The response is: "Happy to help! I'll write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values."

Bottom Screenshot: The same notebook is shown, but the code has been executed. The output in the terminal is:

```
Mean: 30.0  
Minimum: 10  
Maximum: 50
```

The Gemini chat interface remains the same on the right sidebar.

Task 2:

Compare Gemini and Copilot outputs for a Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs

The image displays two development environments side-by-side, comparing the outputs of Gemini and Copilot for a Python function that checks if a number is an Armstrong number.

Top Environment (Google Colab):

- Code Editor:** Contains a Python function `def is_armstrong_number(number):` with a docstring, variable declarations, a loop to calculate the sum of digits raised to the power of the number of digits, and example usage calls: `print(is_armstrong_number(153))`, `print(is_armstrong_number(9))`, and `print(is_armstrong_number(10))`.
- Output:** Shows the results of the function calls: `True`, `True`, and `False`.
- Gemini Chat:** A chat window on the right showing a prompt "generate a Python function that checks whether a number is an Armstrong number" and a response from Gemini: "Sure, I can generate a Python function that checks whether a number is an Armstrong number."

Bottom Environment (VS Code):

- Code Editor:** Contains the same Python function `def is_armstrong_number(n: int) -> bool:` with a docstring, variable declarations, a loop to calculate the sum of digits raised to the power of the number of digits, and example usage calls: `print(is_armstrong_number(153))`, `print(is_armstrong_number(9474))`, and `print(is_armstrong_number(123))`.
- Output:** Shows the results of the function calls: `True`, `True`, and `False`.
- Copilot Chat:** A chat window on the right showing a prompt "generate a Python function that checks whether a number is an Armstrong number" and a response from Copilot: "Sure, I can generate a Python function that checks whether a number is an Armstrong number."

> Used 1 reference

Aspect	Your Code (<code>is_armstrong_number(n)</code>)	Comparison Code (<code>is_armstrong_number(number)</code>)
Parameter Name	<code>n</code>	<code>number</code>
Docstring	Detailed, includes Args and Returns	Brief, single-line
Negative Handling	Uses <code>abs(n)</code> (handles negatives, but not standard for Armstrong)	Uses <code>number</code> directly (no handling for negatives)
Digits Extraction	<code>digits = str(abs(n))</code>	<code>num_str = str(number)</code>
Digits Count	<code>power = len(digits)</code>	<code>num_digits = len(num_str)</code>
Sum Calculation	Generator expression: <code>sum(int(d) ** power for d in digits)</code>	For loop: <code>for digit in num_str: armstrong_sum += ...</code>
Return Statement	<code>return n == total</code>	<code>return armstrong_sum == number</code>
Example Usage	In <code>if __name__ == "__main__":</code> block	Directly after function definition
Readability	More concise, uses Python idioms	More explicit, easier for beginners to follow
Edge Cases	Handles negative input (though not standard for Armstrong numbers)	Does not handle negatives

Observation:
Both functions correctly check for Armstrong numbers, but differ in style and handling of negative inputs. Your code is more concise.

Add Context... AL2.py:1-14

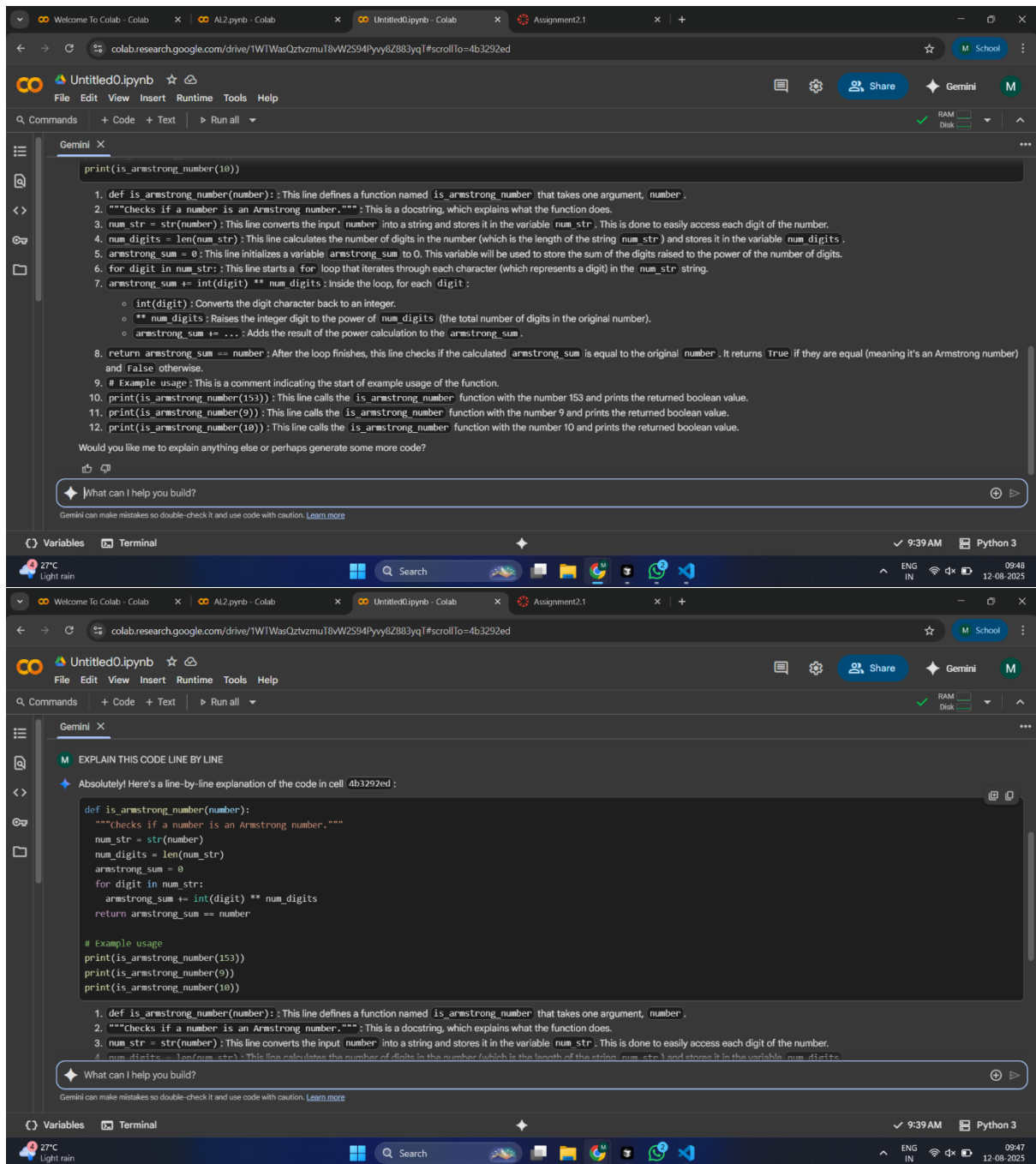
Add context (#), extensions (@), commands (/)

Ask GPT-4.1

Task 3:

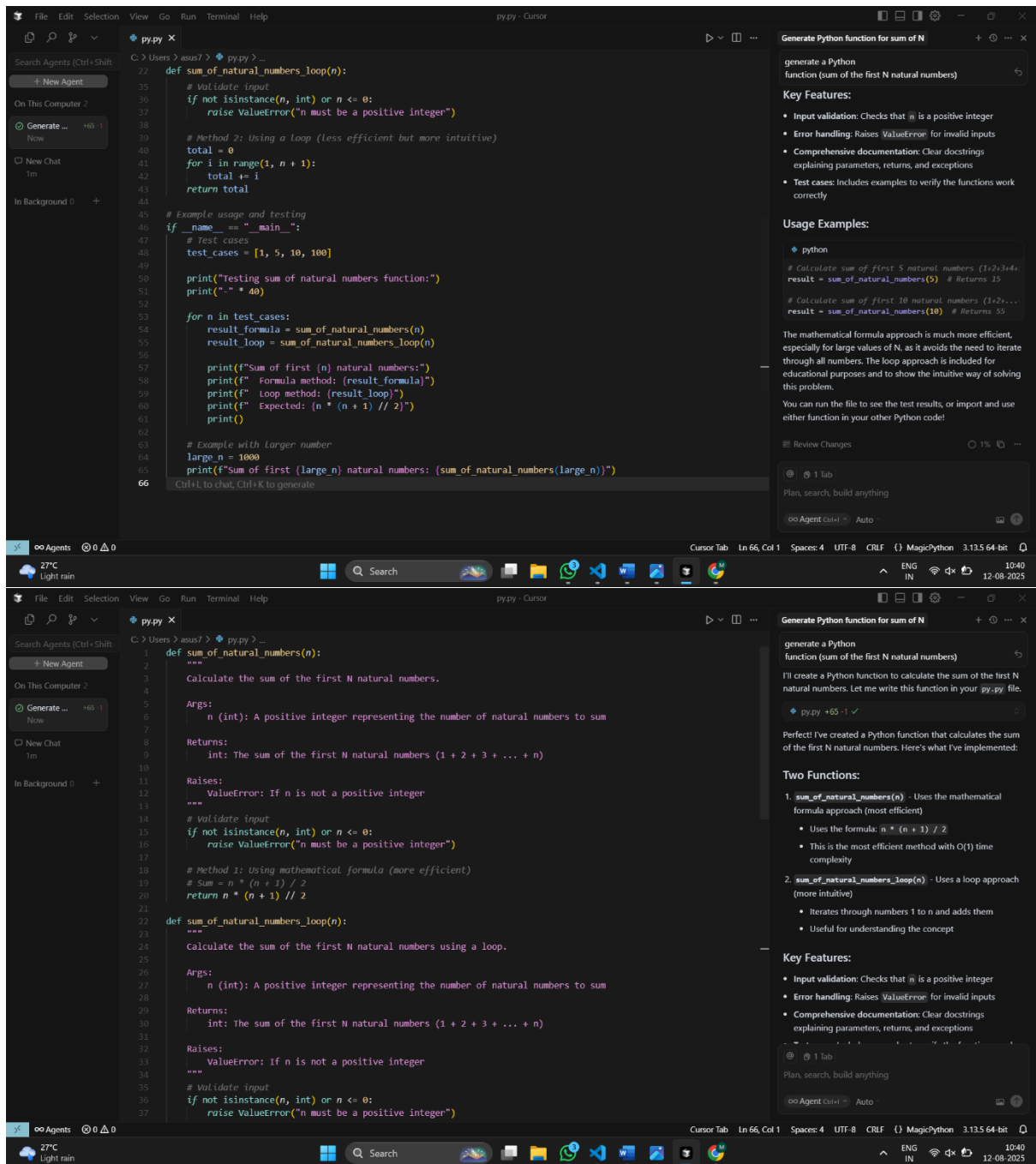
Ask Gemini to explain a Python function (e.g., `is_prime(n)` or `is_palindrome(s)`) line by line.

Choose either a prime-checking or palindrome-checking function and document the explanation provided by Gemini



Task 4:

Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output.



Task5

Students need to write a Python program to calculate the sum of odd numbers and even numbers in a given tuple.

Refactor the code to improve logic and readability

Untitled2.ipynb - Colab

colab.research.google.com/drive/1ezTZwN5v-M7LAHarwGL2TzfXrfvxnLxR#scrollTo=fb317e01

Untitled2.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

0s

"""Calculates the sum of odd and even numbers in a tuple.

Args:
 numbers: A tuple of integers.

Returns:
 A tuple containing the sum of odd numbers and the sum of even numbers.
"""

Use list comprehension for a more concise way to separate odd and even numbers
odd_numbers = [number for number in numbers if number % 2 != 0]
even_numbers = [number for number in numbers if number % 2 == 0]

Use the built-in sum function for clarity
sum_odd = sum(odd_numbers)
sum_even = sum(even_numbers)

return sum_odd, sum_even

Example usage:
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
odd_sum, even_sum = sum_odd_even(my_tuple)

print(f"Tuple: {my_tuple}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")

Tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum of odd numbers: 25
Sum of even numbers: 30