**NAME:SAI KUSHAL**

**ROLL NO:2403A51338**

**BATCH:14**

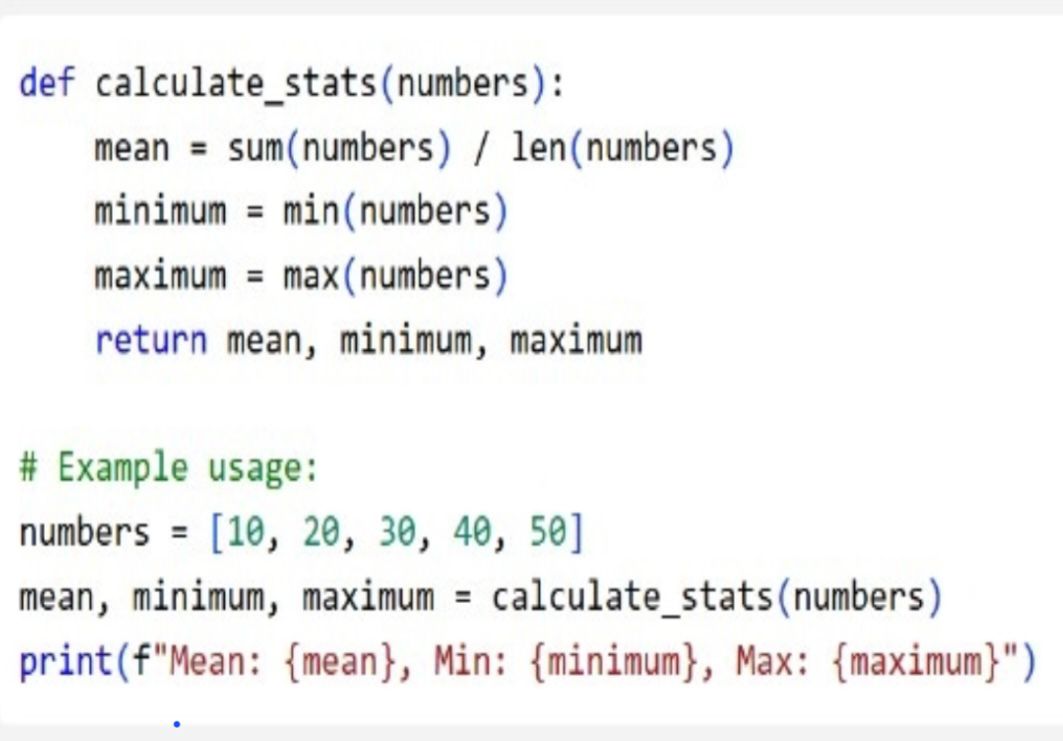
**WEEK:1**

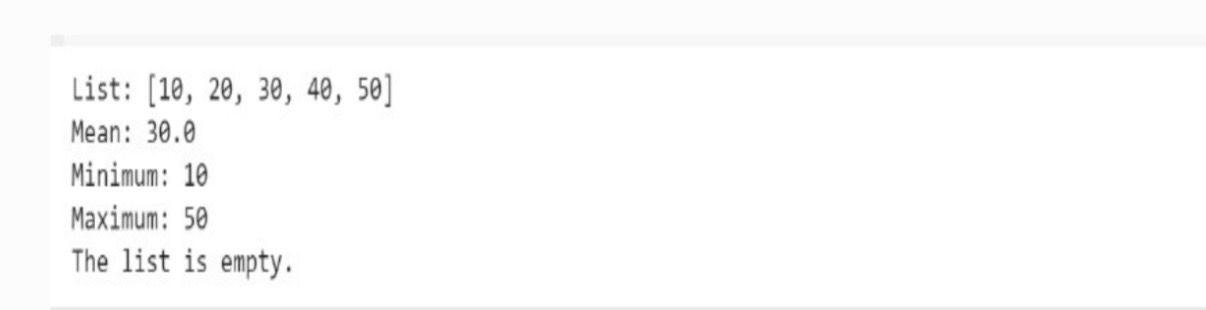
## Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI

# Task Description #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.

# CODE:





# OBSERVATION:

1.The function calculate\_stats(numbers) takes a list of numbers as input.

2. It calculates the mean by dividing the sum of numbers by the count.

3. It finds the minimum using Python’s built-in min() function.

4. It finds the maximum using Python’s built-in max() function.

5. The function returns all three values: mean, minimum, and maximum.

6. In the example, a list [10, 20, 30, 40, 50] is created.

7. This list is passed into the function calculate\_stats(numbers).

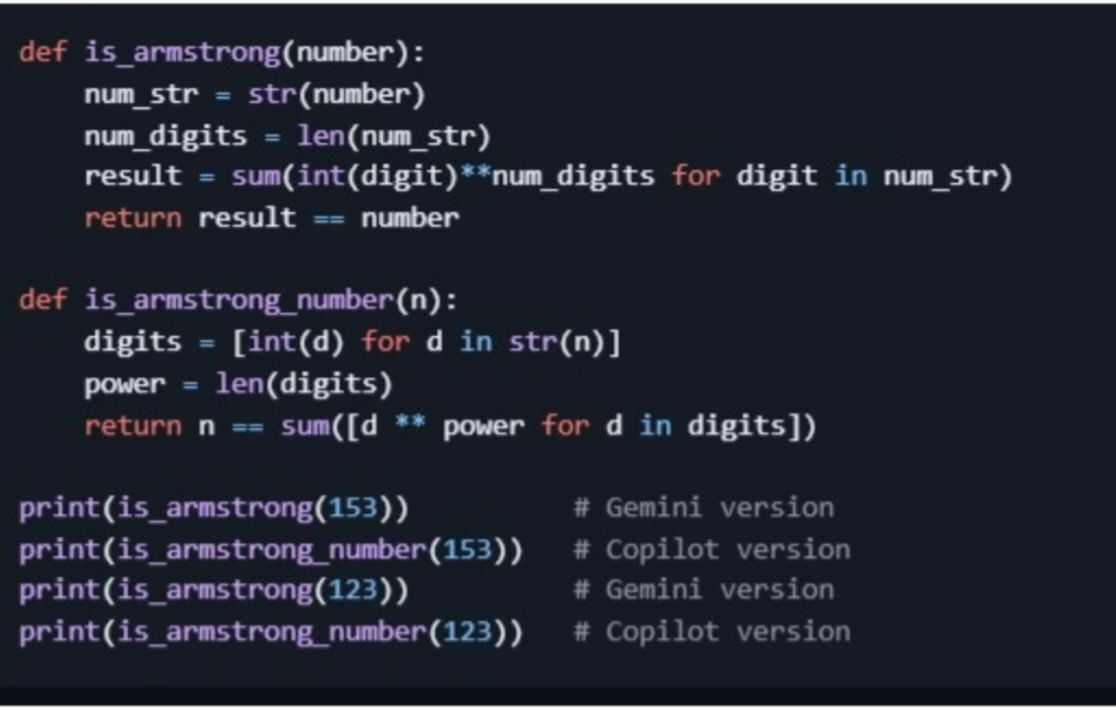
8. The returned values are stored in mean, minimum, maximum.

9. The results are printed using an f-string for formatting.

10. The output shows mean = 30.0, minimum = 10, and maximum = 50.

**Task Description #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.





**OBSERVATION:**

1.An Armstrong number is a number that is equal to the sum of its digits raised to the power of the number of digits.

2.The function is\_armstrong(number) converts the number into a string for easy digit extraction.

3.It counts the digits using len(num\_str).

4.Then, it computes the sum of each digit raised to the power of the number of digits.

5.If this sum equals the original number, it returns True; otherwise, False.

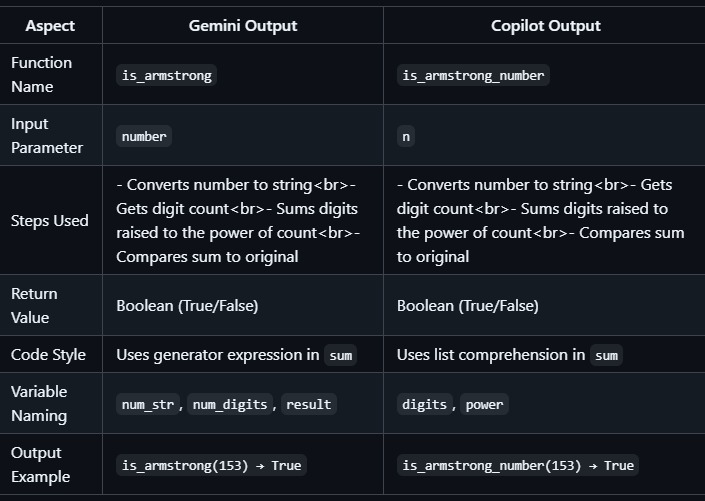
6.The second function is\_armstrong\_number(n) does the same logic but in a slightly different way.

7.It creates a list of digits using list comprehension [int(d) for d in str(n)].

8.The number of digits is stored in power.

9.It checks if n equals the sum of digits raised to power.

10.Example: 153 is Armstrong (1³ + 5³ + 3³ = 153), but 123 is not.

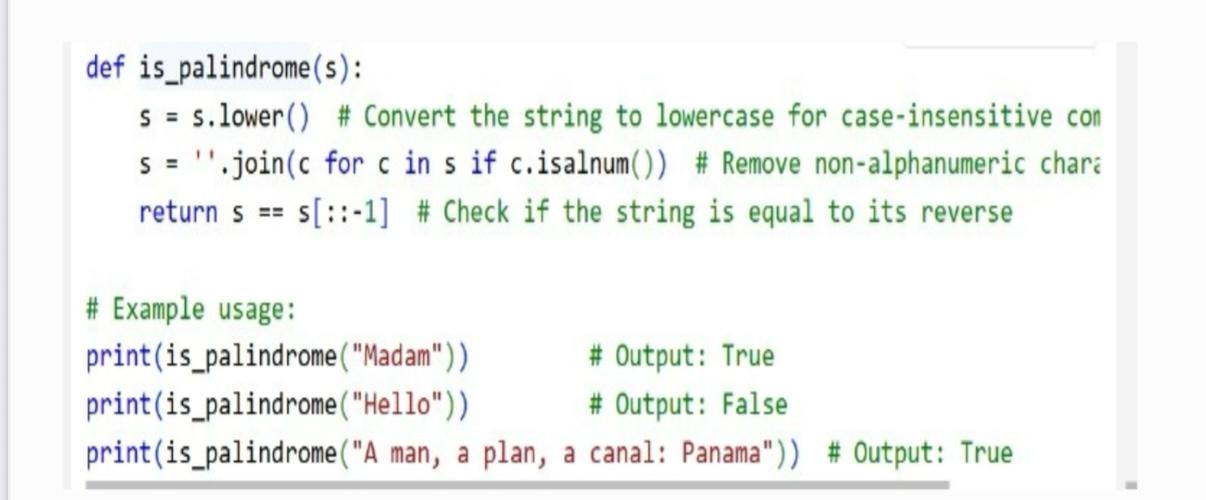


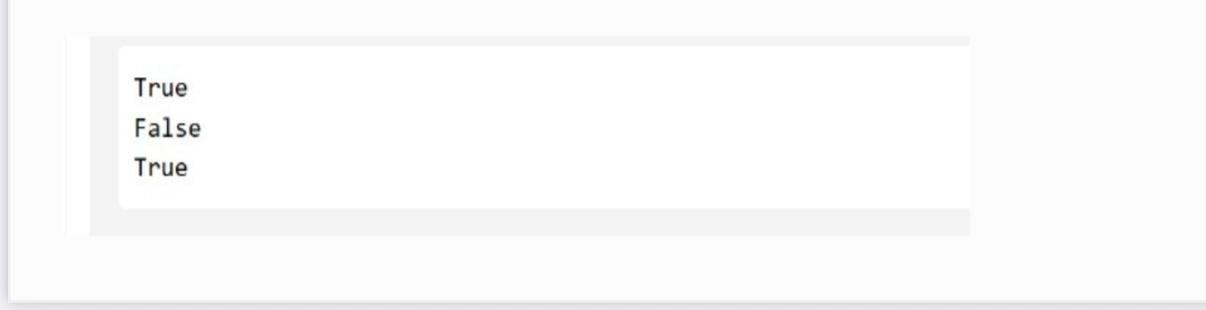
**TASK:3**

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.

# CODE:





# OBSERVATION:

1.The function is\_palindrome(s) checks whether a string is a palindrome.

2.First, s.lower() converts the string to lowercase (case-insensitive).

3.Next, ''.join(c for c in s if c.isalnum()) removes all non-alphanumeric characters (spaces, commas, punctuation).

4.The cleaned string is stored back in s.

5.s[::-1] generates the reverse of the string.

6.The function returns True if the string equals its reverse, otherwise False.

7.Example: "Madam" → "madam" → same as reverse → returns True.

8.Example: "Hello" → "hello" → reverse is "olleh" → returns False.

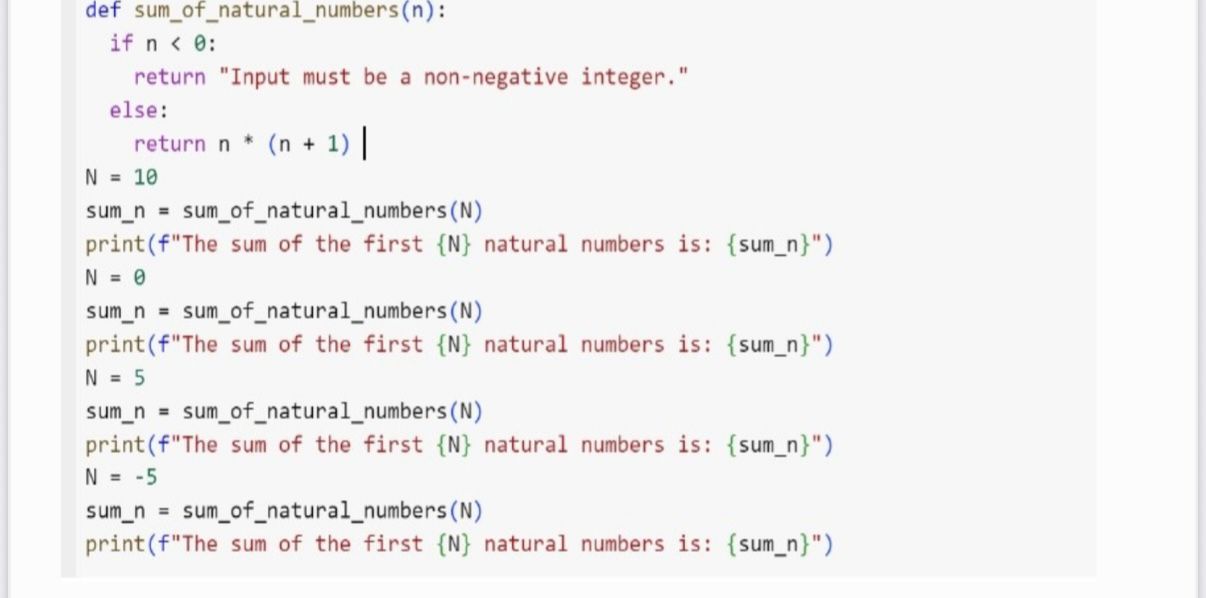
9.Example: "A man, a plan, a canal: Panama" → cleaned to "amanaplanacanalpanama" → palindrome → True.

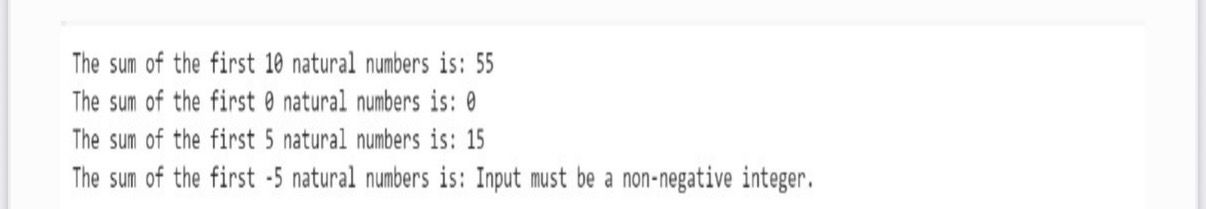
10.Thus, the function works for words, sentences, and ignores case/extra characters.

# TASK:4

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

# CODE:





# OBSERVATION:

1.The function sum\_of\_natural\_numbers(n) calculates the sum of the first n natural numbers.

2.If n < 0, it returns the message: "Input must be a non-negative integer.".

3.Otherwise, it uses the formula n \* (n + 1) // 2 (but here written as n \* (n + 1) / to calculate the sum.

4.This formula works because the sum of the first n natural numbers is well-known.

5. N = 10: the function returns 55, since 1+2+…+10 = 55.

6.N = 0: the sum is 0.

7.N = 5: the sum is 15, since 1+2+3+4+5 = 15.

8.N = -5: since it’s negative, the function returns the error message.

9.The program prints these results one by one.

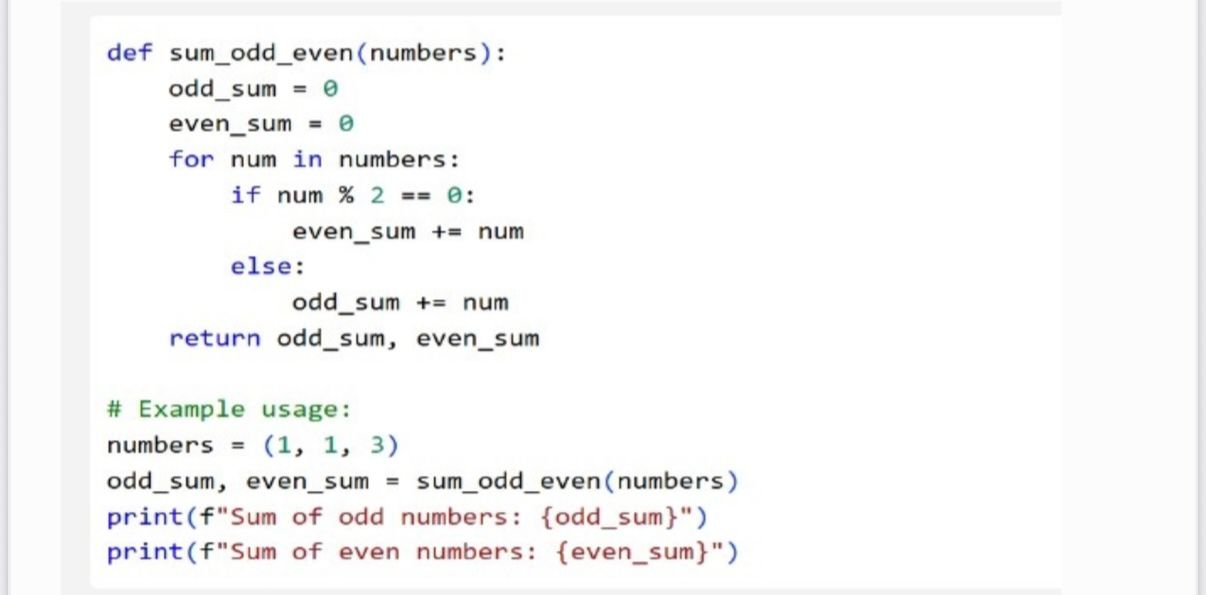
10.Thus, the code checks validity and calculates sums efficiently using a formula.

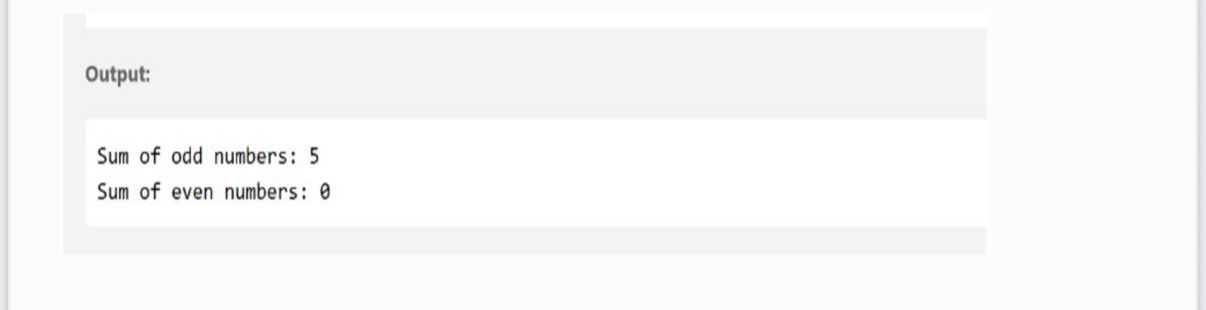
# TASK:5

● 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.

# CODE:





**OBSERVATION:**

**1. The function sum\_odd\_even(numbers) calculates the sum of odd and even numbers separately.**

**2. It starts with two variables: odd\_sum = 0 and even\_sum = 0.**

**3. The function loops through each number in the list numbers.**

**4. Inside the loop, it checks if the number is even (num % 2 == 0).**

**5. If the number is even, it is added to even\_sum.**

**6. Otherwise, the number is odd, and it is added to odd\_sum.**

**7.After the loop, the function returns both sums as (odd\_sum, even\_sum).**

**8. In the example, the input list is (1, 1, 3).**

**9.All numbers are odd, so odd\_sum = 1 + 1 + 3 = 5 and even\_sum = 0.**

**10. The program prints:**

**• “Sum of odd numbers: 5**

**• “Sum of even numbers: 0”**