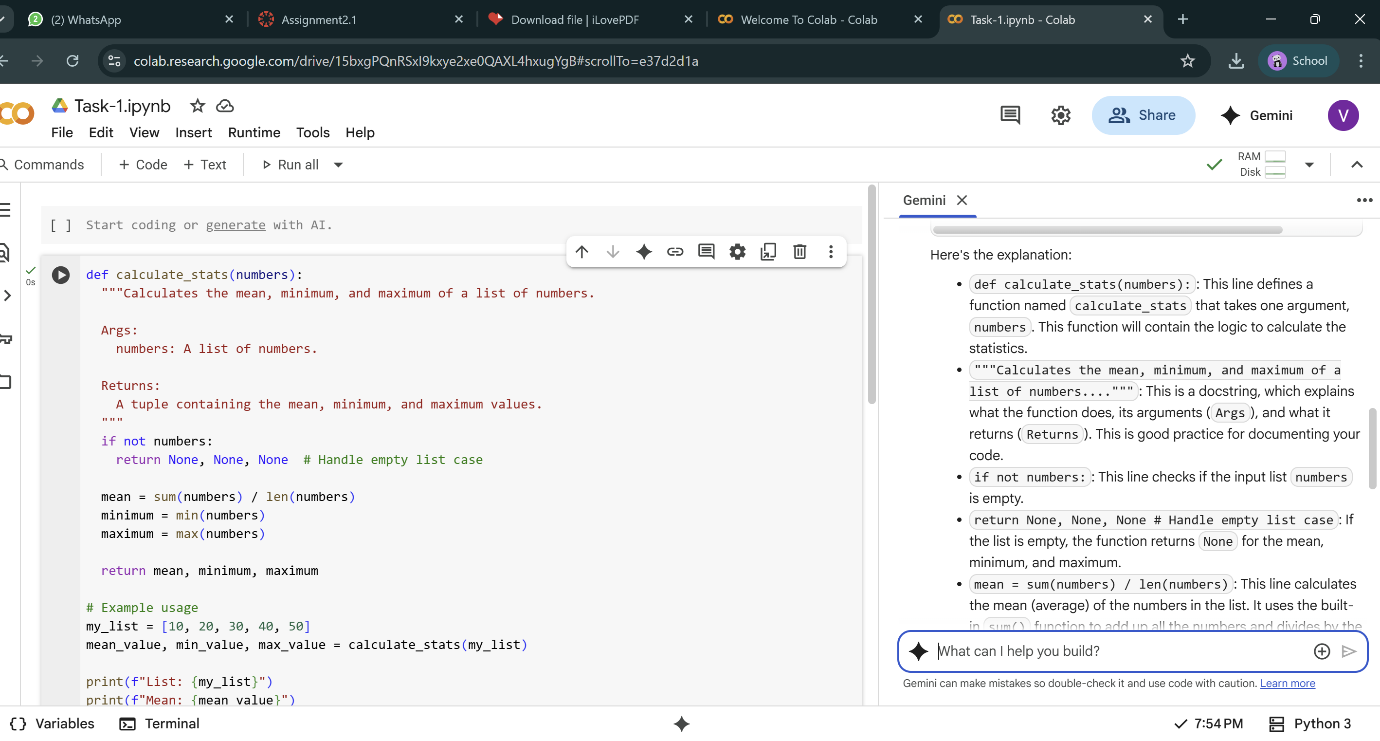
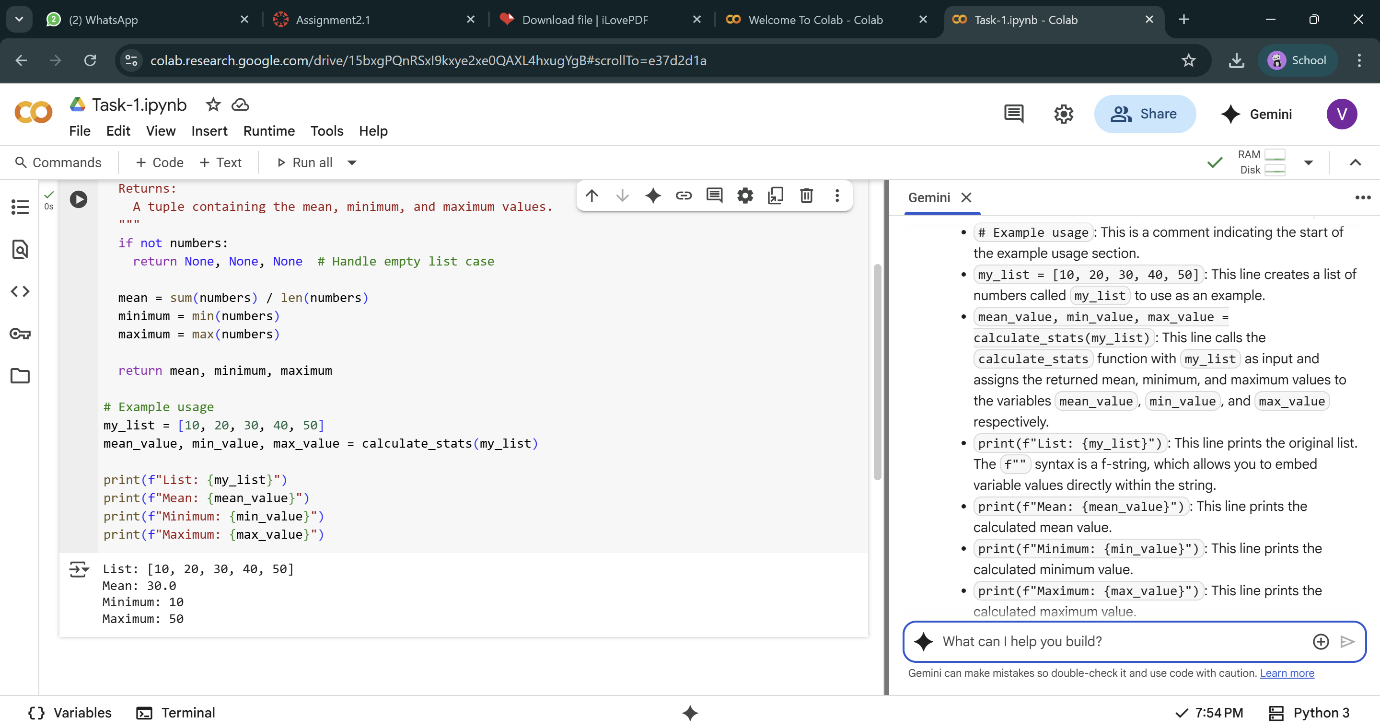
HTNO-2403A52134

Assignment-2

Task-1:

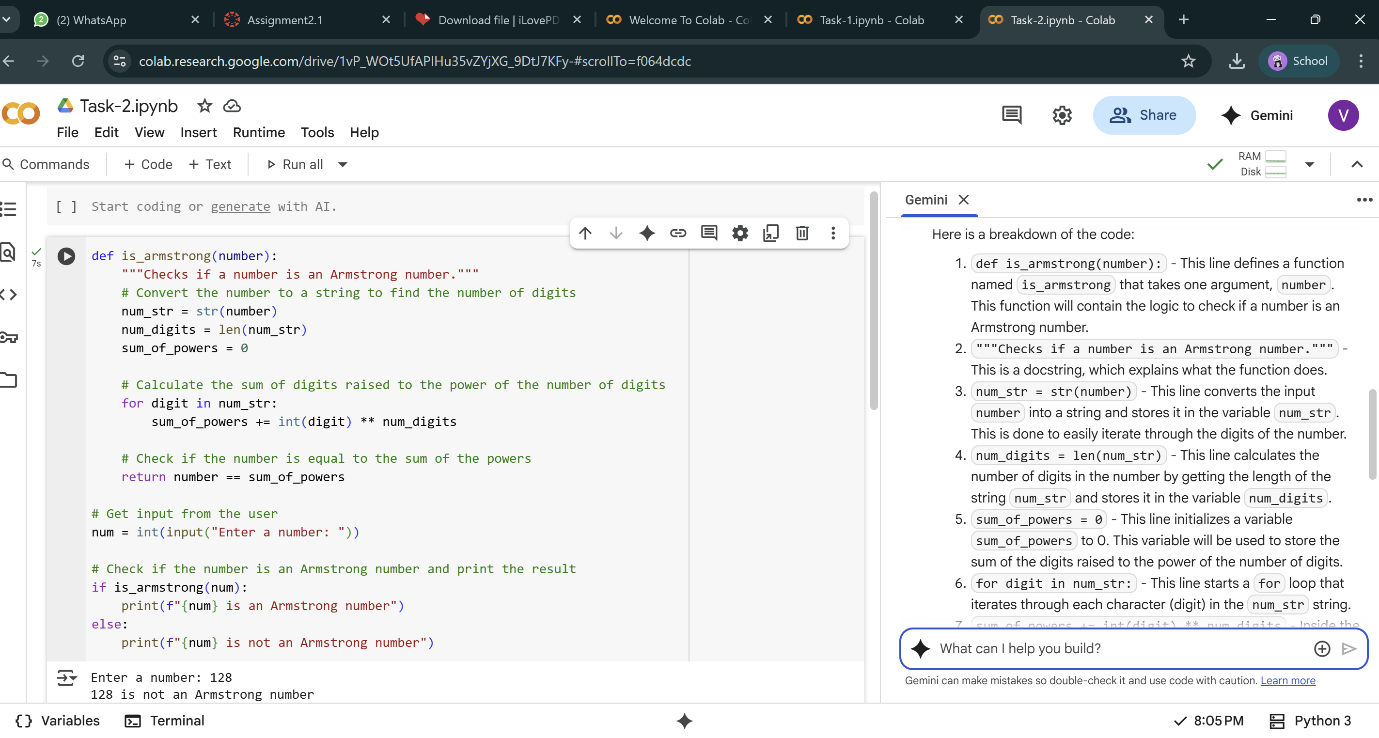
To calculate mean, minimum and maximum values for a list of numbers using Gemini AI in google collab.

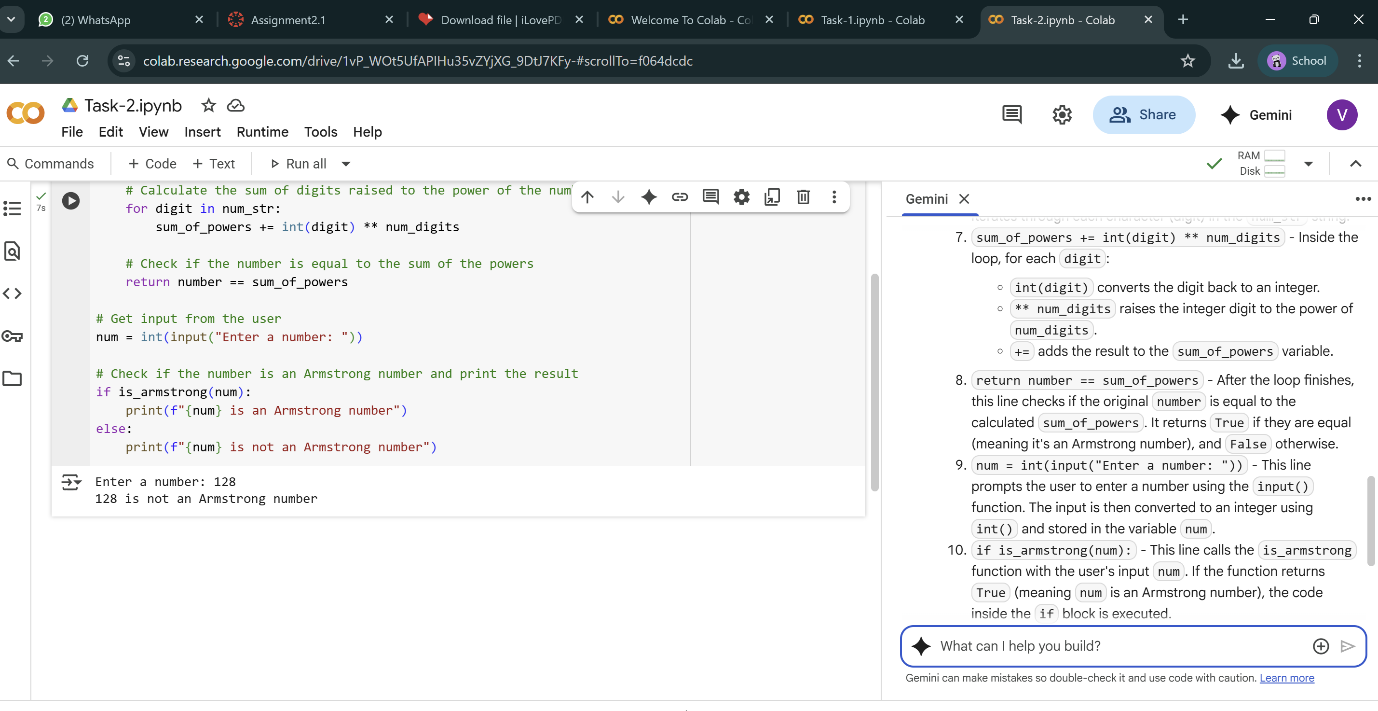


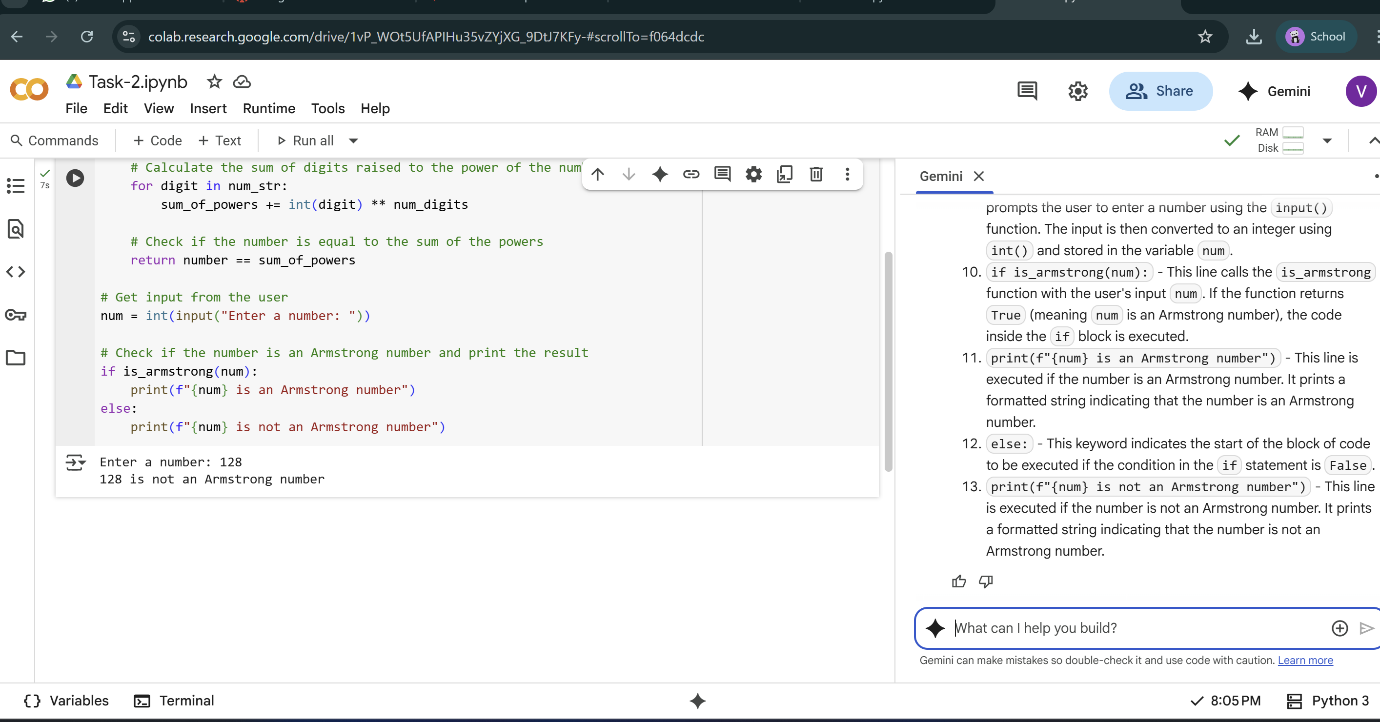


Task-2:

To checks whether a number is an Armstrong number.







Task-3:

1. **Function Definition:**

def is\_prime(number):  
  """  
  Checks if a number is a prime number.  
  
  Args:  
    number: An integer.  
  
  Returns:  
    True if the number is prime, False otherwise.  
  """

* def is\_prime(number): defines a function named is\_prime that takes one argument, number, which is expected to be an integer.
* The docstring explains the function's purpose, arguments, and return value.
  1. **Handling Base Cases:**

# Prime numbers are greater than 1  
if number <= 1:  
  return False

* This if statement handles the basic cases. By definition, prime numbers are integers greater than 1. So, if the input number is less than or equal to 1, the function immediately returns False.
  1. **Checking for Factors:**

# Check for factors from 2 up to the square root of the number  
# We only need to check up to the square root because if a number n has a factor greater than its square root,  
# it must also have a factor less than its square root.  
import math  
for i in range(2, int(math.sqrt(number)) + 1):  
  if number % i == 0:  
    # If a factor is found, it's not a prime number  
    return False

* This is the core logic for checking primality.
* import math: Imports the math module to use the sqrt() function.
* for i in range(2, int(math.sqrt(number)) + 1):: This loop iterates through possible factors starting from 2 up to the integer part of the square root of the input number. We only need to check up to the square root because if a number has a factor larger than its square root, it must also have a corresponding factor smaller than its square root.
* if number % i == 0:: Inside the loop, this checks if the number is divisible by the current value of i (the potential factor). If the remainder of the division is 0, it means i is a factor of number.
* return False: If a factor is found, the number is not prime, so the function immediately returns False.
  1. **Returning True:**

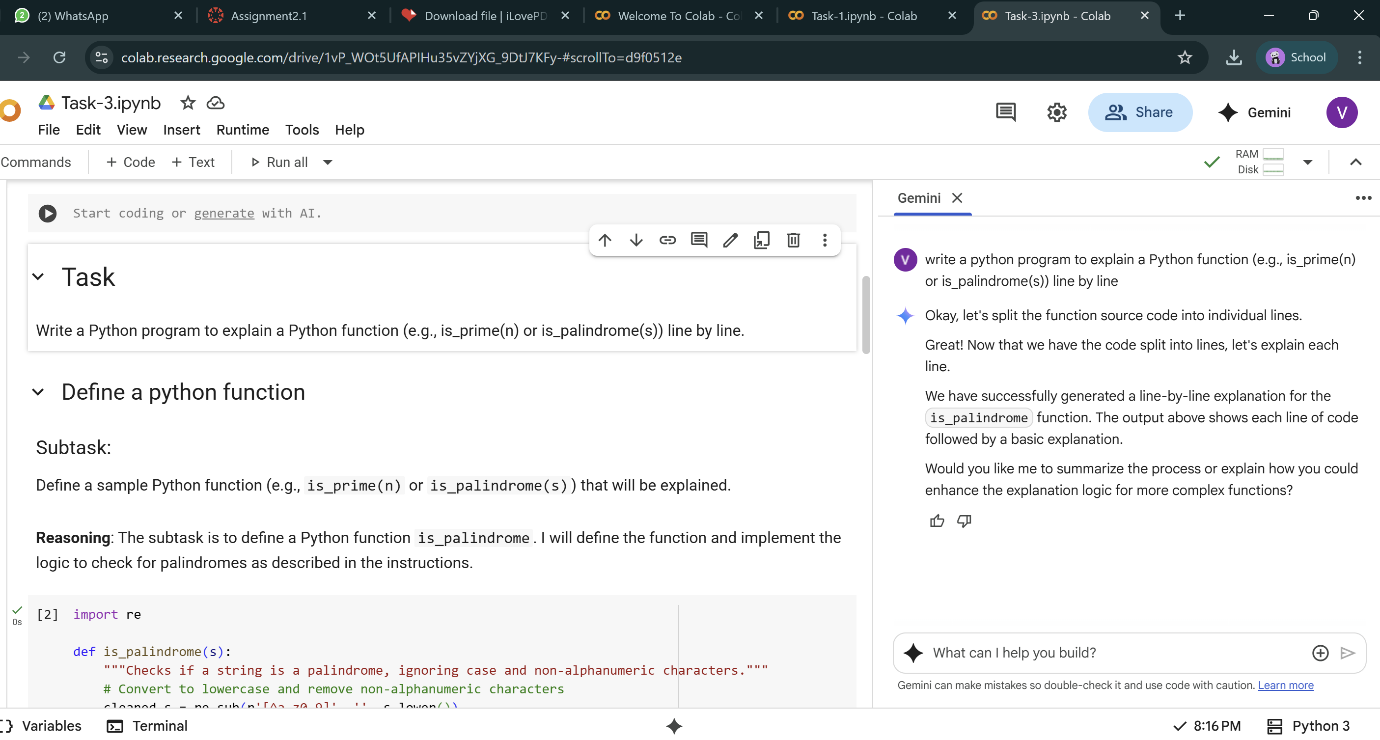
# If no factors are found, it's a prime number  
return True

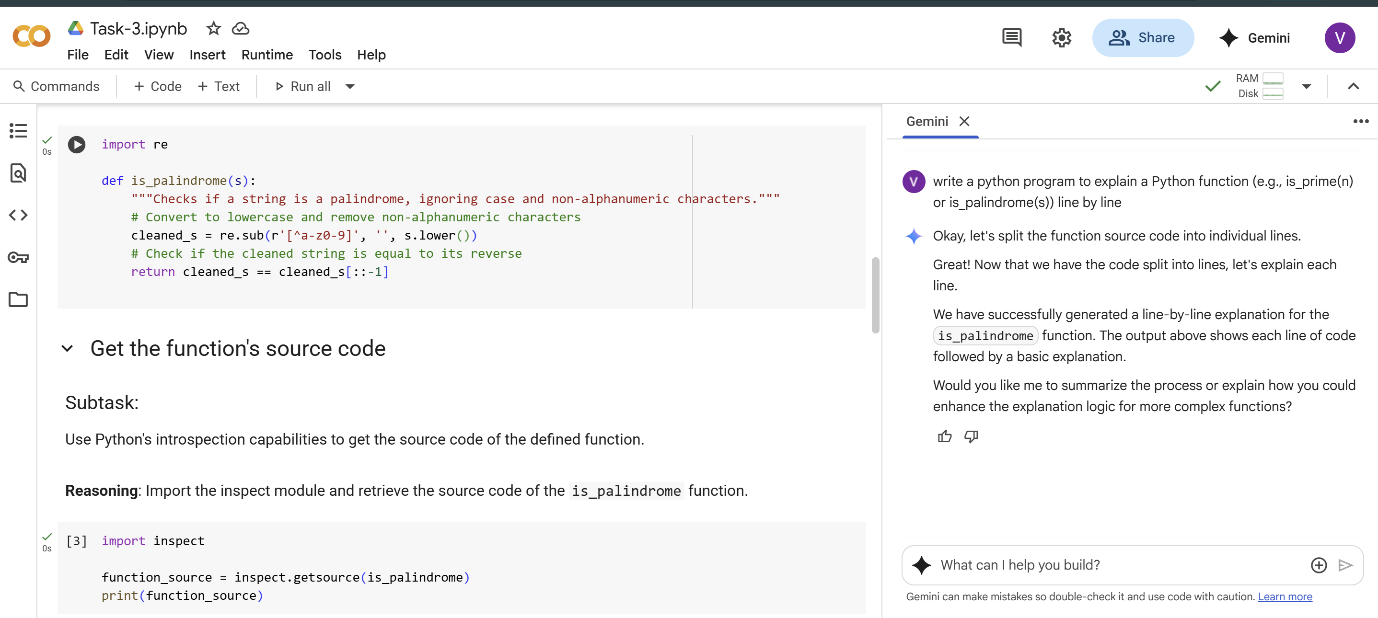
* If the loop completes without finding any factors (meaning the return False statement inside the loop was never executed), it means the number has no factors other than 1 and itself (since we started checking from 2). Therefore, the number is prime, and the function returns True.
  1. **Example Usage:**

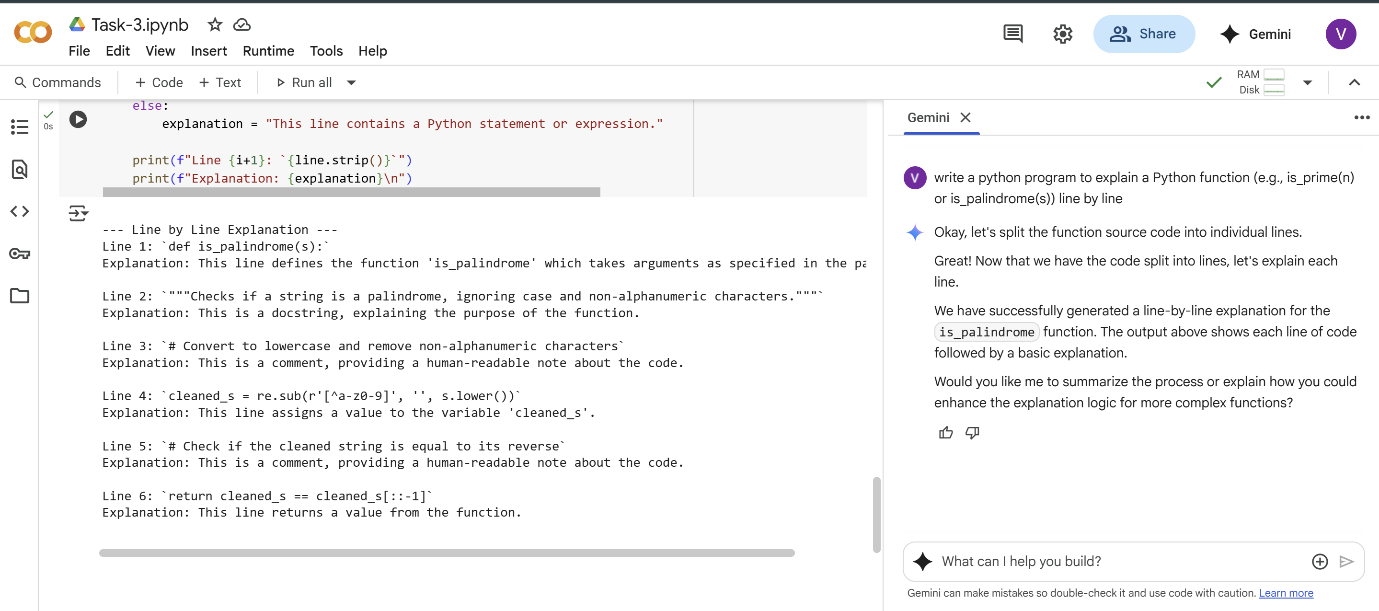
# Example usage:  
num1 = 11  # Prime  
num2 = 4   # Not prime  
num3 = 2   # Prime  
num4 = 17  # Prime  
num5 = 1   # Not prime  
  
print(f"{num1} is a prime number: {is\_prime(num1)}")  
print(f"{num2} is a prime number: {is\_prime(num2)}")  
print(f"{num3} is a prime number: {is\_prime(num3)}")  
print(f"{num4} is a prime number: {is\_prime(num4)}")  
print(f"{num5} is a prime number: {is\_prime(num5)}")

* This section demonstrates how to call the is\_prime function with different numbers and prints the results, showing whether each number is prime or not.

In essence, the is\_prime function efficiently checks for primality by iterating through potential divisors up to the square root of the number.







Task-4:

Compare cursor AI’s generated code with gemini’s output:

Cursor AI’s OUTPUT:

### How it works (step by step)

- \*\*Input and validation\*\*

- The program reads `n` from the user and converts it to `int`.

- The function `sum\_first\_n` checks if `n < 0` and raises `Value\_Error` if so (sum is defined for non-negative integers).

- \*\*Using the formula\*\*

- It computes the sum with the arithmetic series formula: `n \* (n + 1) // 2`.

- Reason: First term = 1, last term = `n`, number of terms = `n`, average term = `(1 + n) / 2`, so sum = `n \* (1 + n) / 2`.

- `//` is integer division, ensuring the result is an integer.

- \*\*Output\*\*

- The result from `sum\_first\_n(n)` is printed.

### Alternative (loop approach)

- Initialize `total = 0`.

- For each `i` from 1 to `n`, do `total += i`.

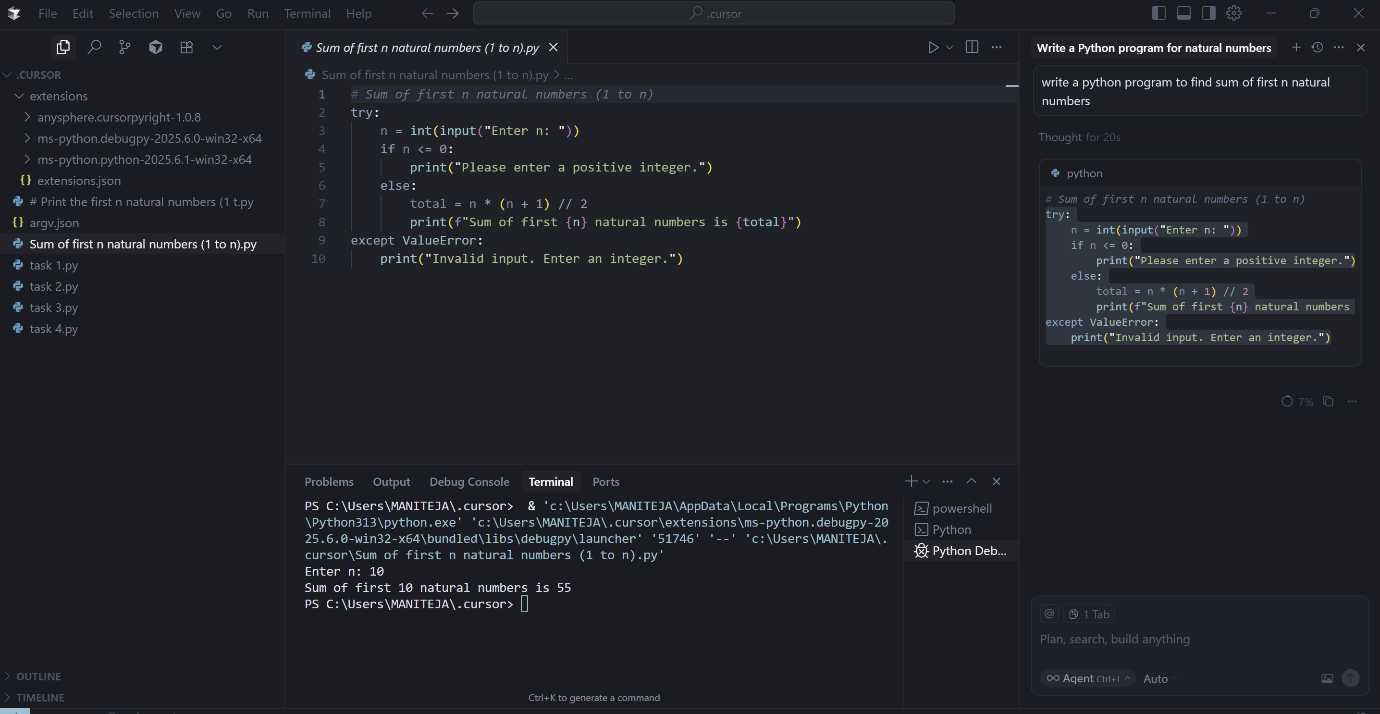
- Print `total`. This is O(n); the formula is O(1).

### Example

- If `n = 5`, sum = `5 \* 6 // 2 = 15`.

- Edited code uses a math formula with input validation; prints the sum.

- Formula is constant time and mathematically derived from the average of first and last terms.



Gemini’s Output:

**1.Function Definition:**

def sum\_of\_first\_n\_natural\_numbers(n):  
  """  
  Calculates the sum of the first n natural numbers.  
  
  Args:  
    n: An integer representing the number of natural numbers.  
  
  Returns:  
    The sum of the first n natural numbers.  
    Returns 0 if n is less than or equal to 0.  
  """

* def sum\_of\_first\_n\_natural\_numbers(n): defines a function named sum\_of\_first\_n\_natural\_numbers that takes one argument, n, which is expected to be an integer representing the count of natural numbers.
* The docstring explains the function's purpose, arguments, and return value.
  1. **Handling Non-positive Input:**

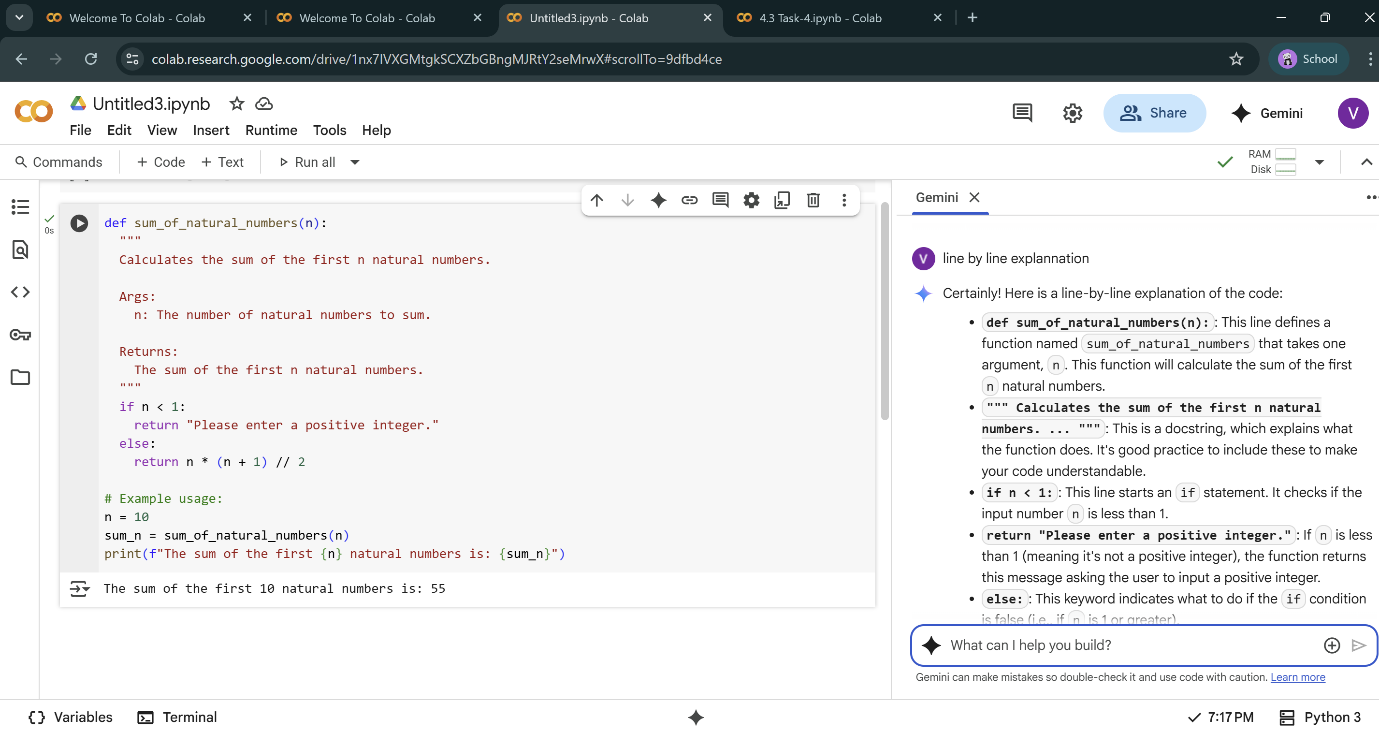
if n <= 0:  
  return 0

* This if statement checks if the input n is less than or equal to 0. Natural numbers are usually considered positive integers (1, 2, 3, ...). If n is not a positive integer, the sum of the first 'n' natural numbers is considered to be 0, so the function returns 0.
  1. **Calculating the Sum using a Formula:**

else:  
  # Using the formula: sum of first n natural numbers = n \* (n + 1) / 2  
  return n \* (n + 1) // 2

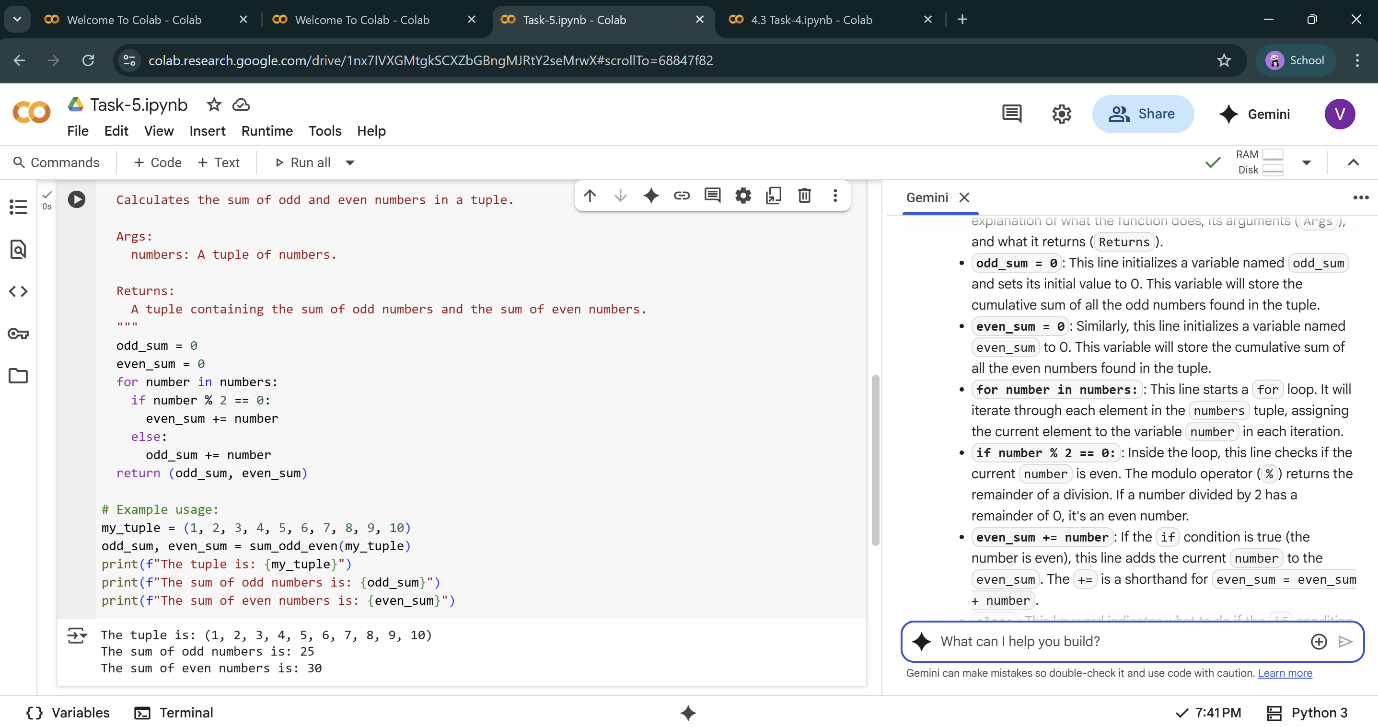
* If n is a positive integer, the code calculates the sum using the well-known formula for the sum of an arithmetic series: n \* (n + 1) / 2.
* return n \* (n + 1) // 2: The result of the calculation is returned. The // operator is used for integer division, which is appropriate here since the sum of natural numbers will always be an integer.
  1. **Example Usage:**
  2. # Example usage:  
     n1 = 10  
     sum1 = sum\_of\_first\_n\_natural\_numbers(n1)  
     print(f"The sum of the first {n1} natural numbers is: {sum1}")  
       
     n2 = 0  
     sum2 = sum\_of\_first\_n\_natural\_numbers(n2)  
     print(f"The sum of the first {n2} natural numbers is: {sum2}")  
     n3 = 5  
     sum3 = sum\_of\_first\_n\_natural\_numbers(n3)  
     print(f"The sum of the first {n3} natural numbers is: {sum3}")
* This section demonstrates how to use the function with different values of n (10, 0, and 5) and prints the calculated sums.

In summary, the function efficiently calculates the sum of the first 'n' natural numbers using a mathematical formula and handles the case where 'n' is not a positive integer.



Task-5:

To calculate the sum of odd numbers and even numbers in a given tuple.



Explanation of the code line by line:

* **def sum\_odd\_even(numbers):**: This line defines a function named sum\_odd\_even that takes one argument, numbers. This function is designed to calculate the sum of odd and even numbers within the input numbers (which is expected to be a tuple).
* **""" Calculates the sum of odd and even numbers in a tuple. ... """**: This is a docstring, providing a brief explanation of what the function does, its arguments (Args), and what it returns (Returns).
* **odd\_sum = 0**: This line initializes a variable named odd\_sum and sets its initial value to 0. This variable will store the cumulative sum of all the odd numbers found in the tuple.
* **even\_sum = 0**: Similarly, this line initializes a variable named even\_sum to 0. This variable will store the cumulative sum of all the even numbers found in the tuple.
* **for number in numbers:**: This line starts a for loop. It will iterate through each element in the numbers tuple, assigning the current element to the variable number in each iteration.
* **if number % 2 == 0:**: Inside the loop, this line checks if the current number is even. The modulo operator (%) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it's an even number.
* **even\_sum += number**: If the if condition is true (the number is even), this line adds the current number to the even\_sum. The += is a shorthand for even\_sum = even\_sum + number.
* **else:**: This keyword indicates what to do if the if condition is false (i.e., if the number is not even, it's odd).
* **odd\_sum += number**: If the number is odd, this line adds the current number to the odd\_sum. The += is a shorthand for odd\_sum = odd\_sum + number.
* **return (odd\_sum, even\_sum)**: After the loop has finished iterating through all the numbers in the tuple, this line returns a tuple containing the final odd\_sum and even\_sum.
* **# Example usage:**: This is a comment indicating the following lines are an example of how to use the function.
* **my\_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)**: This line creates a tuple named my\_tuple with some example numbers.
* **odd\_sum, even\_sum = sum\_odd\_even(my\_tuple)**: This line calls the sum\_odd\_even function with my\_tuple as the argument. The function returns a tuple containing the odd and even sums, which are then unpacked into the odd\_sum and even\_sum variables.
* **print(f"The tuple is: {my\_tuple}")**: This line prints the original tuple.
* **print(f"The sum of odd numbers is: {odd\_sum}")**: This line prints the calculated sum of odd numbers.
* **print(f"The sum of even numbers is: {even\_sum}")**: This line prints the calculated sum of even numbers.