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.

Expected Output #1

* Functional code with correct output and screenshot.

def calculate\_stats(numbers):

  """

  Calculates the mean, minimum, and maximum of a list of numbers.

  Args:

    numbers: A list of numbers.

  Returns:

    A tuple containing the mean, minimum, and maximum values.

    Returns (None, None, None) if the input list is empty.

  """

  if not numbers:

    return None, None, None

  mean\_value = sum(numbers) / len(numbers)

  min\_value = min(numbers)

  max\_value = max(numbers)

  return mean\_value, min\_value, max\_value

# Example usage:

my\_list = [10, 20, 30, 40, 50]

mean, min\_val, max\_val = calculate\_stats(my\_list)

if mean is not None:

  print(f"List: {my\_list}")

  print(f"Mean: {mean}")

  print(f"Minimum: {min\_val}")

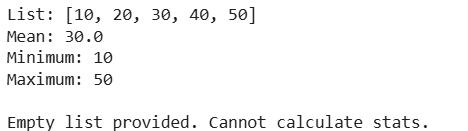
  print(f"Maximum: {max\_val}")

empty\_list = []

mean, min\_val, max\_val = calculate\_stats(empty\_list)

if mean is None:

  print("\nEmpty list provided. Cannot calculate stats

output: 

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.

Expected Output #2

* Side-by-side comparison table with observations and screenshots.

**PROMPT:** Write a Python function called is\_armstrong that takes a positive integer n as input and returns True if it's an Armstrong number and otherwise False. The function should work for numbers with any number of digits.

**GEMINI:**

**CODE:**

def is\_armstrong(n: int) -> bool:

    """ Checks if a positive integer is an Armstrong number.

    An Armstrong number is an n-digit number that is equal to the sum

    of its own digits, each raised to the power of n.

    Args:

        n: A positive integer.

    Returns:

        True if n is an Armstrong number, False otherwise. """

    if not isinstance(n, int) or n < 0:

        # Handle non-integer or negative inputs gracefully

        print(f"Warning: Input '{n}' must be a positive integer. Returning False.")

        return False

    # Convert the number to a string to easily get its length (number of digits)

    num\_str = str(n)

    num\_digits = len(num\_str)

    sum\_of\_powers = 0

    # Iterate through each digit in the string representation of the number

    for digit\_char in num\_str:

        digit = int(digit\_char)  # Convert the character digit back to an integer

        sum\_of\_powers += digit \*\* num\_digits  # Add the digit raised to the power of num\_digits

    # Compare the calculated sum with the original number

    return sum\_of\_powers == n

**OUTPUT:**

print("Checking Armstrong Numbers:")

print(f"Is 5 an Armstrong number? {is\_armstrong(5)}")        # Expected: True (5^1 = 5)

print(f"Is 153 an Armstrong number? {is\_armstrong(153)}")    # Expected: True (1^3 + 5^3 + 3^3 = 153)

print(f"Is 9474 an Armstrong number? {is\_armstrong(9474)}")  # Expected: True (9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 + 2401 + 256 = 9474)

**COPILOT:**

**CODE:**

def is\_armstrong(n=None):

# Ask user for input if n is not provided

if n is None:

try:

n = int(input("Enter a positive integer: "))

if n <= 0:

print("Please enter a positive integer.")

return False

except ValueError:

print("Invalid input. Please enter a valid integer.")

return False

# Convert number to string to easily iterate over digits

digits = str(n)

num\_digits = len(digits)

# Calculate the sum of digits raised to the power of num\_digits

total = sum(int(digit) \*\* num\_digits for digit in digits)

# Check if the number is equal to the calculated total

return total == n

**OUT PUT:**

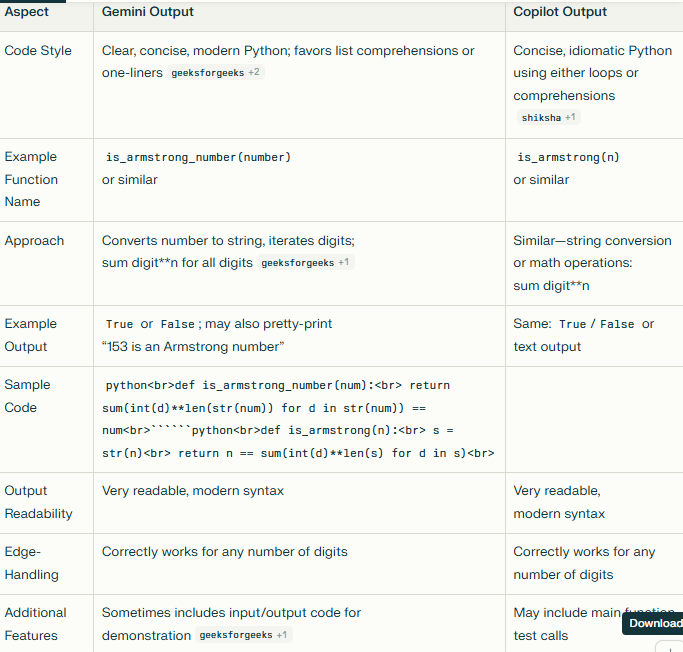
print(is\_armstrong(153)) # True

print(is\_armstrong(9474)) # True

print(is\_armstrong(123)) # False

print(is\_armstrong()) # Prompts user for input

**comparision between Copilot and Gemini:**



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

**CODE:**def is\_prime(n):

if n <= 1:

return False

i = 2

while i \* i <= n:

if n % i == 0:

return False

i += 1

return True

You didn't provide a specific function to explain, so I'll explain a common one: is\_prime(n). This function checks if a given integer n is a prime number.

**The is\_prime(n) Function**

Python

def is\_prime(n):

if n <= 1:

return False

i = 2

while i \* i <= n:

if n % i == 0:

return False

i += 1

return True

**Line-by-Line Explanation**

1. def is\_prime(n):
   * This line defines a function named is\_prime that takes one argument, n. The argument n is the integer we want to check for primality.
2. if n <= 1:
   * This is the first conditional check. **Prime numbers are defined as integers greater than 1**. Therefore, any number less than or equal to 1 (including negative numbers and 0) cannot be prime.
3. return False
   * If the condition n <= 1 is true, the function immediately stops and returns False, indicating that the number is not prime.
4. i = 2
   * We initialize a variable i to 2. This variable will be used as a potential divisor. We start with 2 because it's the smallest prime number and we've already handled the cases for numbers less than 2.
5. while i \* i <= n:
   * This line begins a while loop. The condition i \* i <= n is a key optimization. We only need to check for divisors up to the square root of n. If n has a divisor greater than its square root, it must also have a corresponding divisor that is smaller than its square root. For example, if n=100, and we find the divisor 20, the corresponding divisor is 5 (100/20=5), which is less than the square root of 100 (which is 10). By only checking up to i \* i <= n (or equivalently i <= sqrt(n)), we significantly reduce the number of checks needed.
6. if n % i == 0:
   * Inside the loop, this line checks if n is evenly divisible by i. The modulo operator % returns the remainder of the division. If the remainder is 0, it means i is a divisor of n.
7. return False
   * If n is found to be divisible by i, it means n has a divisor other than 1 and itself, so it cannot be prime. The function immediately returns False.
8. i += 1
   * This line increments the value of i by 1. The loop then continues to the next iteration with the new value of i, checking the next potential divisor.
9. return True
   * If the loop completes without finding any divisors (i.e., the function never returned False), it means that n has no divisors other than 1 and itself within the range we checked. Therefore, the number is prime, and the function returns True.

Task Description #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.
* Optionally, compare Cursor AI’s generated code with Gemini’s output.

**Cursor:**

**CODE:**

def sum\_natural\_numbers(*n*):

    """ Calculate the sum of the first N natural numbers.

    Args:

        n (int): A positive integer representing the count of natural numbers

    Returns:

        int: The sum of numbers from 1 to n

    Formula: sum = n \* (n + 1) / 2 """

*if* n <= 0:

*return* 0

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

def test\_sum\_function():

    """Test the sum\_natural\_numbers function with various inputs."""

*# Test cases*

    test\_cases = [

        (1, 1),      *# sum of first 1 number: 1*

        (2, 3),      *# sum of first 2 numbers: 1 + 2 = 3*

        (3, 6),      *# sum of first 3 numbers: 1 + 2 + 3 = 6*

        (5, 15),     *# sum of first 5 numbers: 1 + 2 + 3 + 4 + 5 = 15*

        (10, 55),    *# sum of first 10 numbers: 1 + 2 + ... + 10 = 55*

        (100, 5050), *# sum of first 100 numbers: 1 + 2 + ... + 100 = 5050*

        (0, 0),      *# edge case: 0*

        (-5, 0),     *# edge case: negative number*]

    print("Testing sum\_natural\_numbers function:")

    print("=" \* 50)

*for* n, expected *in* test\_cases:

        result = sum\_natural\_numbers(n)

        status = "✓ PASS" *if* result == expected *else* "✗ FAIL"

        print(f"n = {n:3d} | Expected: {expected:5d} | Got: {result:5d} | {status}")

    print("=" \* 50)

*# Additional verification with manual calculation*

    print("\nManual verification:")

  n = 7

    manual\_sum = 1 + 2 + 3 + 4 + 5 + 6 + 7

    function\_sum = sum\_natural\_numbers(n)

    print(f"Manual calculation for n={n}: 1+2+3+4+5+6+7 = {manual\_sum}")

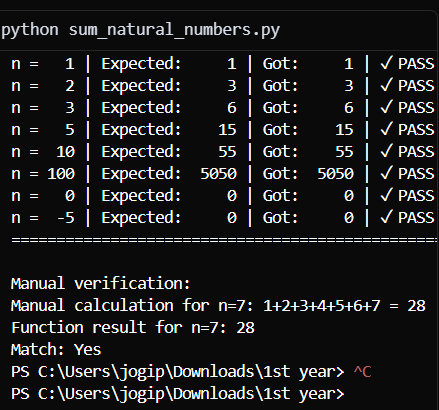
    print(f"Function result for n={n}: {function\_sum}")

    print(f"Match: {'Yes' *if* manual\_sum == function\_sum *else* 'No'}")

*if* \_\_name\_\_ == "\_\_main\_\_":

    test\_sum\_function()

**OUT PUT:**



**Gemini ai:  
CODE:**

**def sum\_natural\_numbers(n):**

**"""**

**Calculates the sum of the first n natural numbers.**

**Assumes n is a non-negative integer.**

**"""**

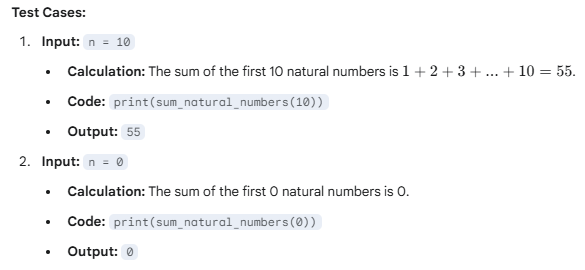
**if n < 0:**

**return "Input must be a non-negative integer."**

**# Using the formula for the sum of an arithmetic series**

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

**OUT PUT:**



**Task Description #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:**

**tu= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

**even = 0**

**odd = 0**

**for i in tu:**

**if i % 2 == 0:**

**even += i**

**else:**

**odd += i**

**print("SUM OF EVEN tuple:", even)**

**print("SUM OF ODD tuple:", odd)**

**OUT PUT:**

