ASSINGMENT-2

Task-1

def analyze numbers(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 = sum(numbers) / len(numbers)

minimum = min(numbers)

maximum = max(numbers)

return mean, minimum, maximum

# Example usage:

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

Mean \_val, min\_val, max\_val = analyze\_numbers(my\_list)

print(f"List: {my\_list}")

print(f"Mean: {mean\_val}")

print(f"Minimum: {min\_val}")

print(f"Maximum: {max\_val}")

empty\_list = []

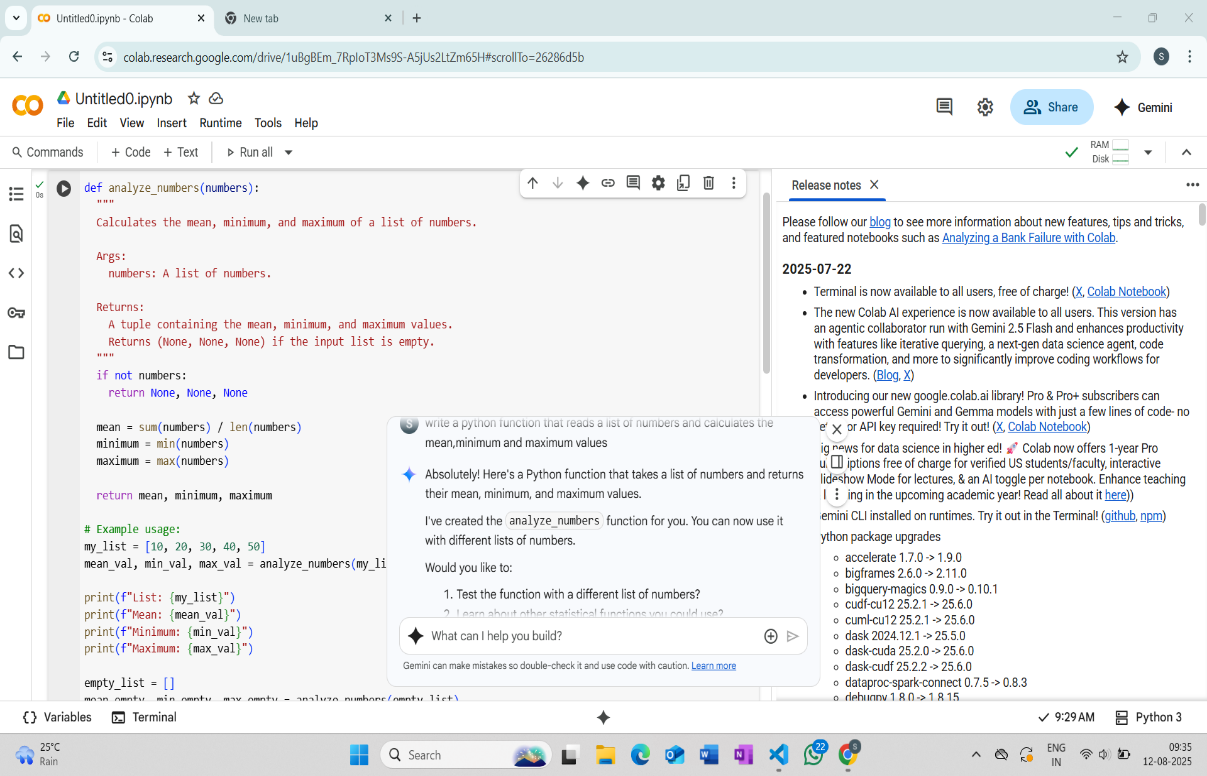
mean\_empty, min\_empty, max\_empty = analyze\_numbers(empty\_list)

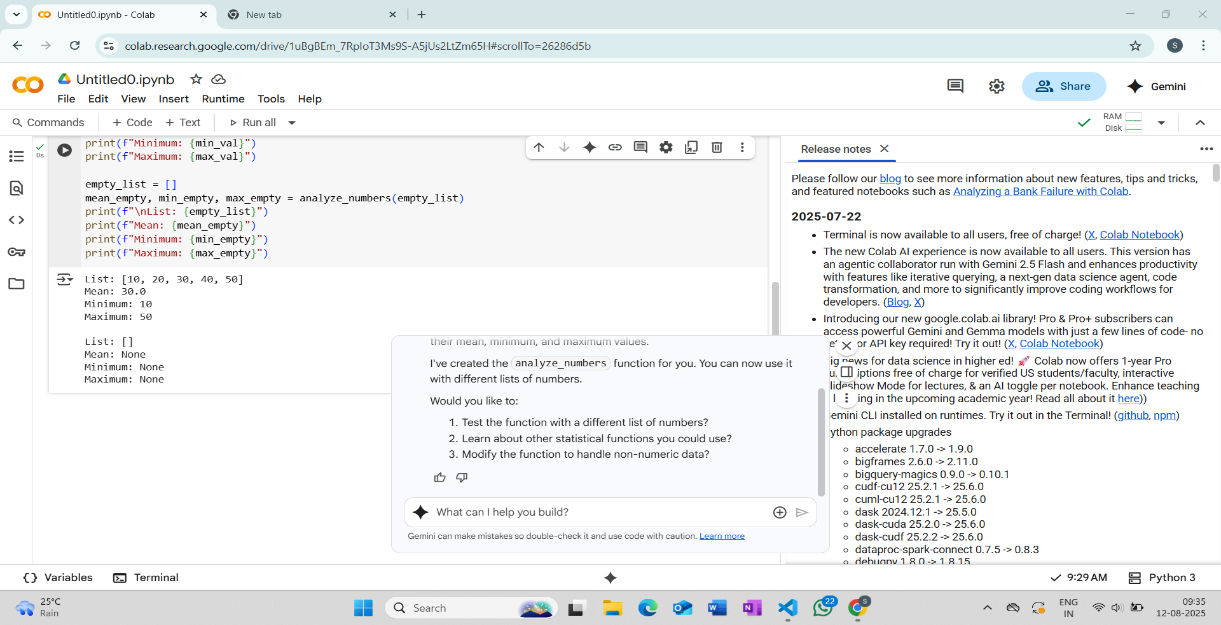
print(f"\nList: {empty\_list}")

print(f"Mean: {mean\_empty}")

print(f"Minimum: {min\_empty}")

print(f"Maximum: {max\_empty}")





Task-2

def is\_armstrong(number):

# Convert the number to a string to easily iterate over digits

digits = str(number)

num\_digits = len(digits)

# Calculate the sum of each digit raised to the power of num\_digits

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

# Check if the total equals the original number

return total == number

# Example usage

print(is\_armstrong(153)) # True

print(is\_armstrong(9474) # True

print(is\_armstrong(123)) # False

This is a code from Gemini-AI:

def is\_armstrong(number):

# Convert the number to a string to easily iterate over digits

digits = str(number)

num\_digits = len(digits)

# Calculate the sum of each digit raised to the power of num\_digits

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

# Check if the total equals the original number

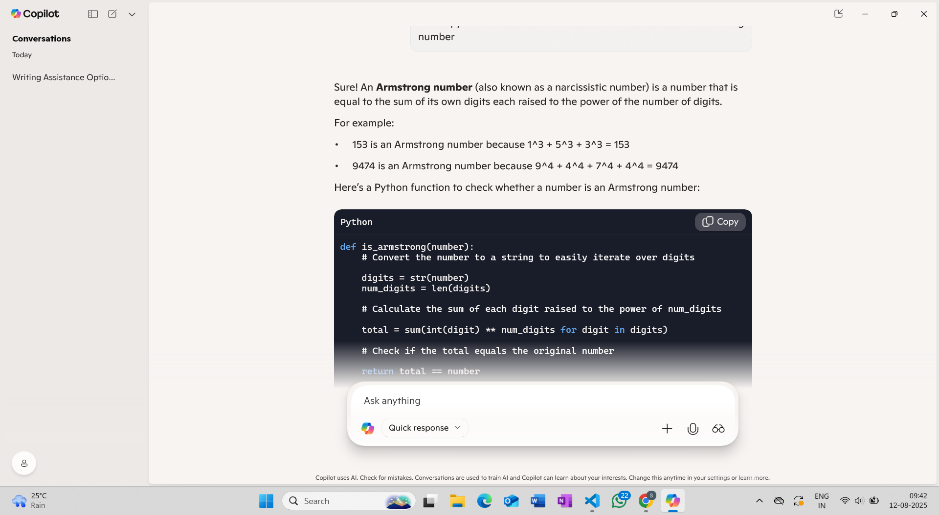
return total == number

# Example usage

print(is\_armstrong(153)) # True

print(is\_armstrong(9474)) # True

print(is\_armstrong(123)) # False



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`.

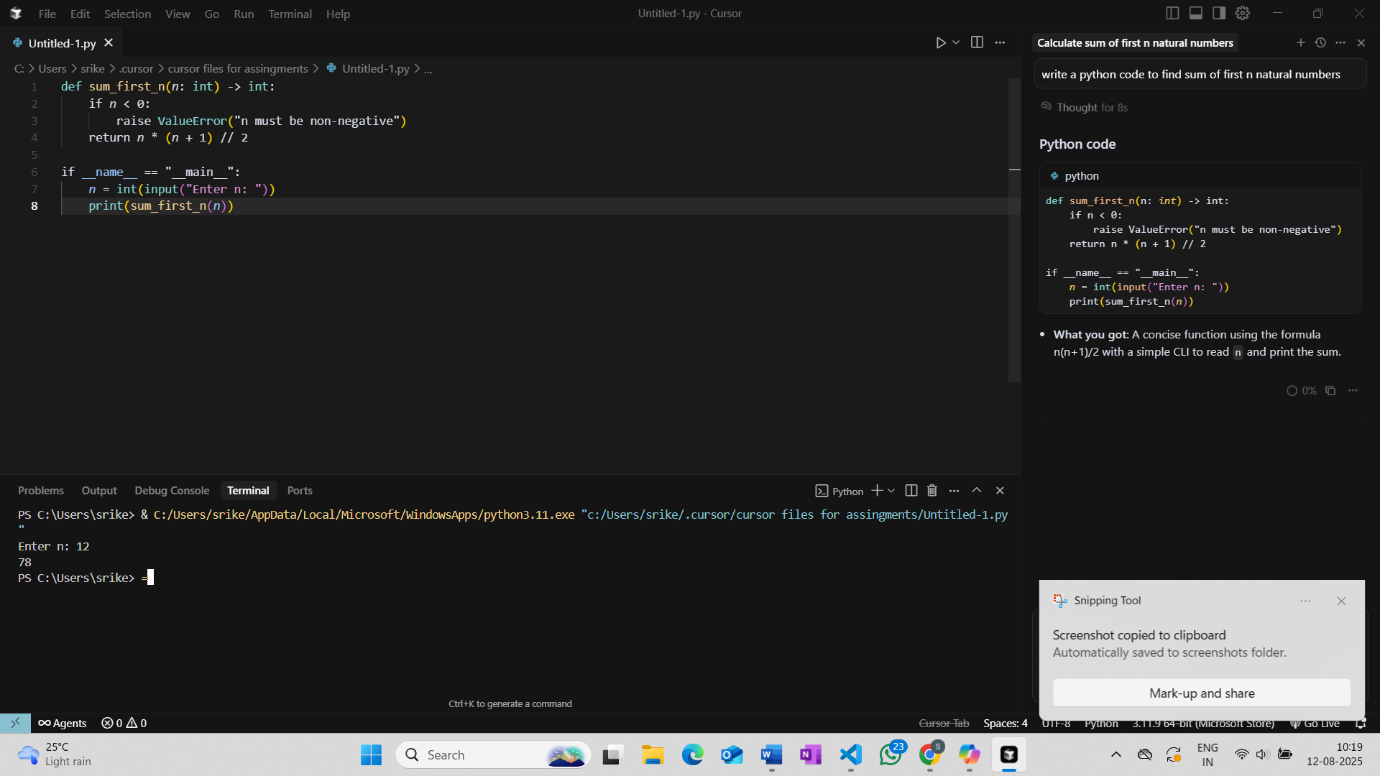
- Print `total`. This is O(n); the formula is O(1).

### 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:**

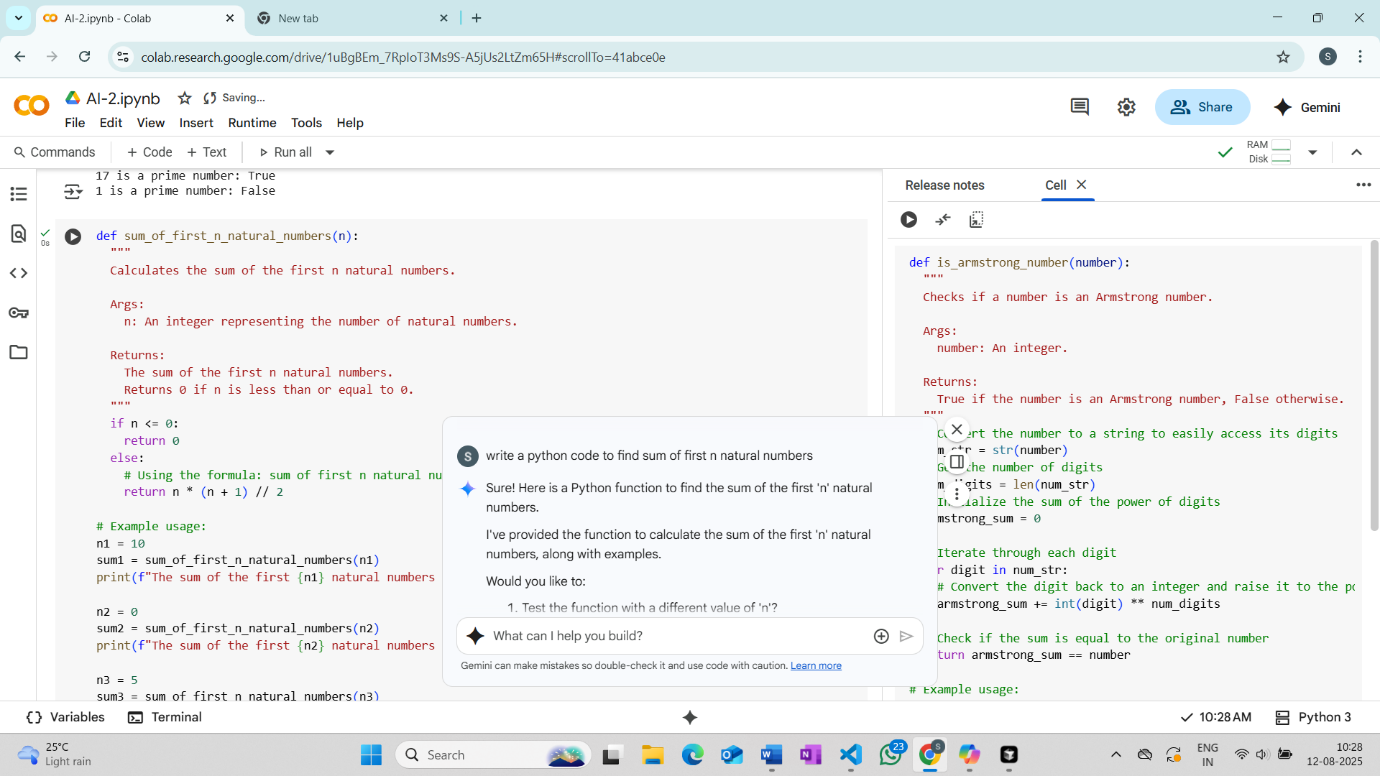
if n <= 0:  
  return 0

* 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:**

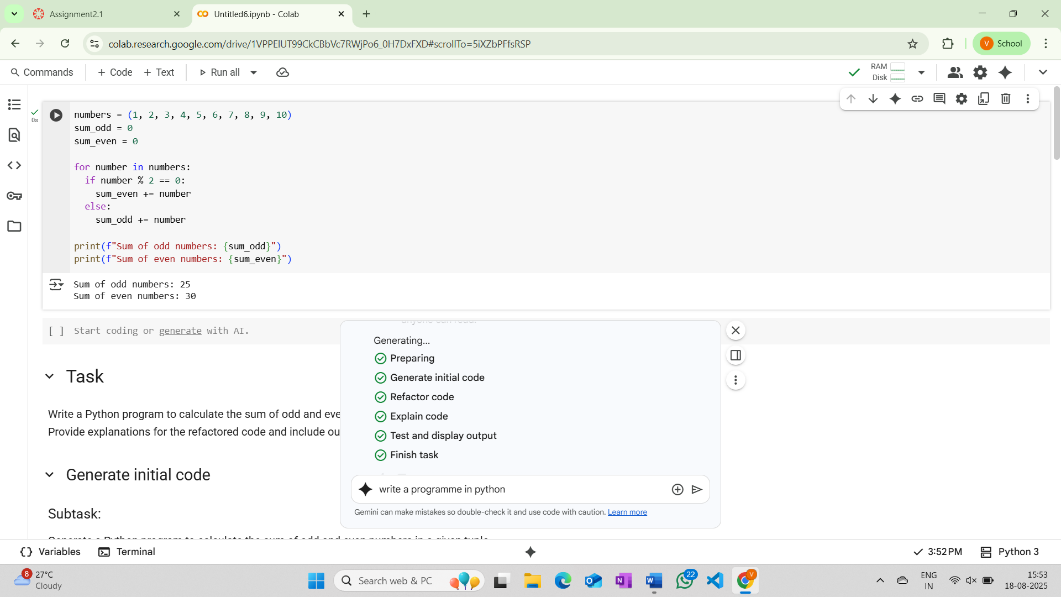
else:  
  # Using the formula: sum of first n natural numbers = n \* (n + 1) / 2  
  return n \* (n + 1) // 2

* 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



m\_odd = sum(number for number in numbers if number % 2 != 0)

This line calculates the sum of odd numbers.

* (number for number in numbers if number % 2 != 0) is a generator expression that iterates through each number in the numbers tuple. The if number % 2 != 0 condition filters out the numbers that are not divisible by 2 (i.e., the odd numbers).
* sum(...) is a built-in Python function that calculates the sum of the elements generated by the expression.

sum\_even = sum(number for number in numbers if number % 2 == 0)

This line calculates the sum of even numbers.

* (number for number in numbers if number % 2 == 0) is a generator expression that iterates through each number in the numbers tuple. The if number % 2 == 0 condition filters out the numbers that are divisible by 2 (i.e., the even numbers).
* sum(...) is a built-in Python function that calculates the sum of the elements generated by the expression.

print(f"Sum of odd numbers: {sum\_odd}")

This line prints the calculated sum of odd numbers.

* print() is a built-in Python function that outputs text to the console.
* f"Sum of odd numbers: {sum\_odd}" is an f-string, which allows you to embed the value of the sum\_odd variable directly within the string.

print(f"Sum of even numbers: {sum\_even}")

This line prints the calculated sum of even numbers.

* print() is a built-in Python function that outputs text to the console.
* f"Sum of even numbers: {sum\_even}" is an f-string, which allows you to embed the value of the sum\_even variable directly within the string.