AI ASSITED CODING

ASSIGNMENT 2.1

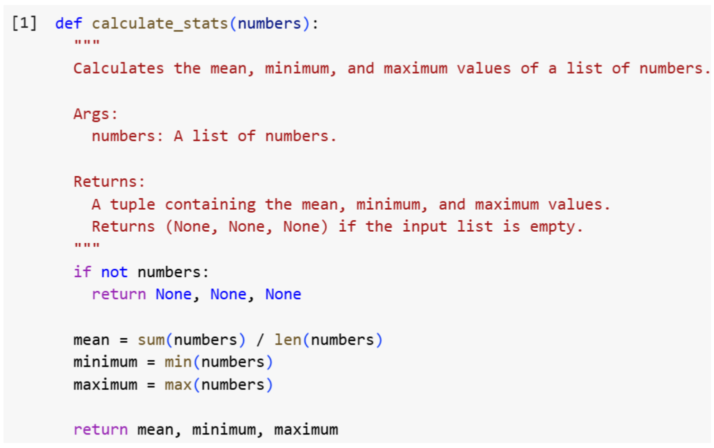
NAME:K.SUPRIYA

ROLL NO:2403A52395

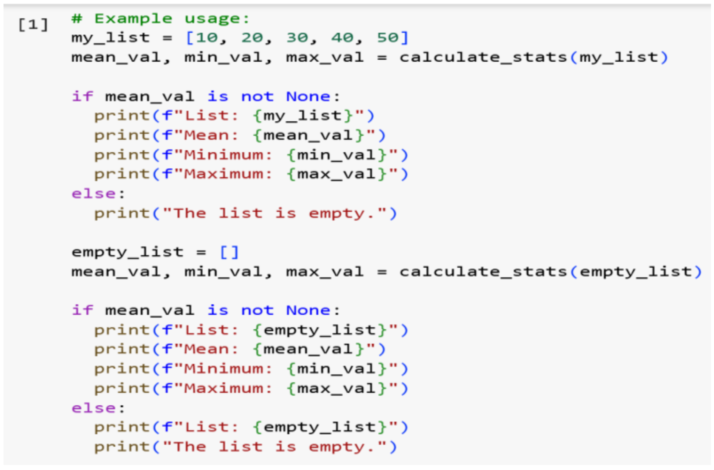
BATCH:14

Task Description #1

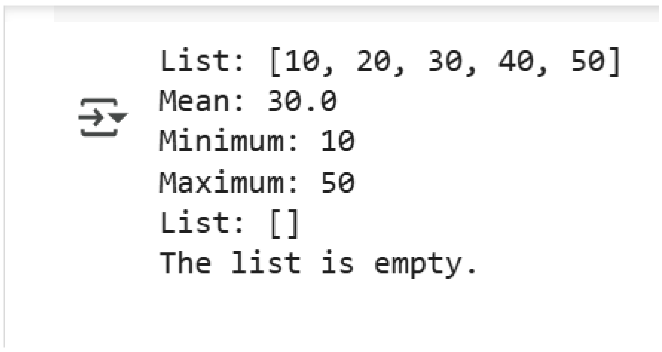
● Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values



```
[1] def calculate_stats(numbers):
      """
      Calculates the mean, minimum, and maximum values 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
```



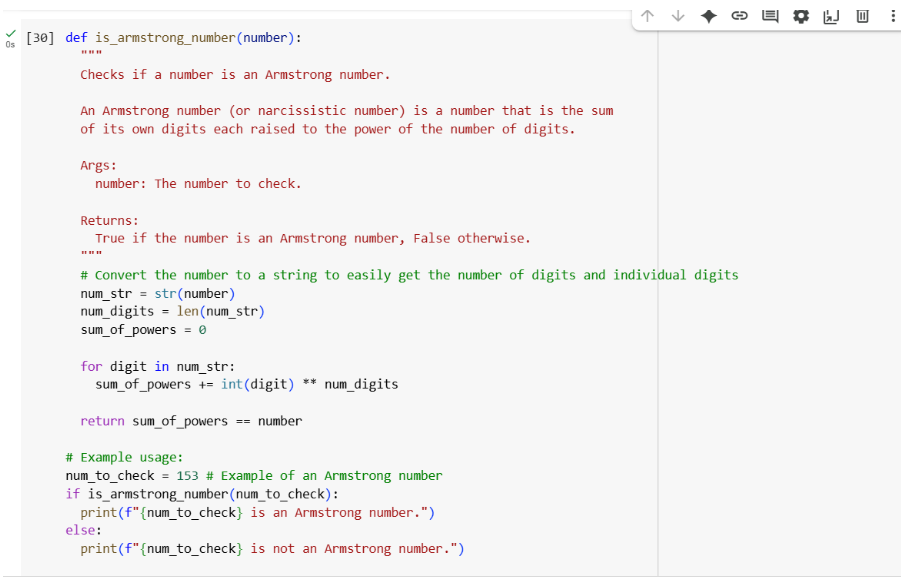
```
[1] # Example usage:
    my_list = [10, 20, 30, 40, 50]
    mean_val, min_val, max_val = calculate_stats(my_list)

    if mean_val is not None:
      print(f"List: {my_list}")
      print(f"Mean: {mean_val}")
      print(f"Minimum: {min_val}")
      print(f"Maximum: {max_val}")
    else:
      print("The list is empty.")

    empty_list = []
    mean_val, min_val, max_val = calculate_stats(empty_list)

    if mean_val is not None:
      print(f"List: {empty_list}")
      print(f"Mean: {mean_val}")
      print(f"Minimum: {min_val}")
      print(f"Maximum: {max_val}")
    else:
      print(f"List: {empty_list}")
      print("The list is empty.")
```



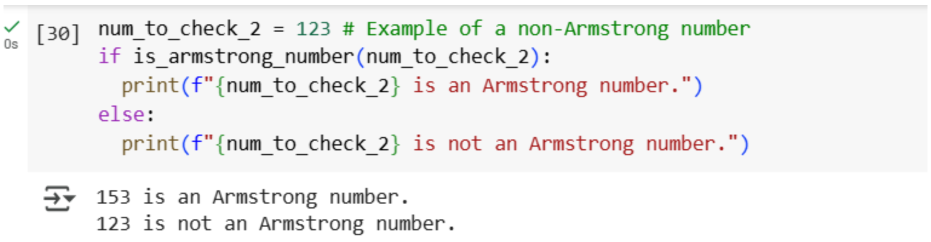
```
List: [10, 20, 30, 40, 50]
Mean: 30.0
Minimum: 10
Maximum: 50
List: []
The list is empty.
```

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.



```
[30] def is_armstrong_number(number):
       """
       Checks if a number is an Armstrong number.

       An Armstrong number (or narcissistic number) is a number that is the sum
       of its own digits each raised to the power of the number of digits.

       Args:
         number: The number to check.

       Returns:
         True if the number is an Armstrong number, False otherwise.
       """
       # Convert the number to a string to easily get the number of digits and individual digits
       num_str = str(number)
       num_digits = len(num_str)
       sum_of_powers = 0

       for digit in num_str:
         sum_of_powers += int(digit) ** num_digits

       return sum_of_powers == number

     # Example usage:
     num_to_check = 153 # Example of an Armstrong number
     if is_armstrong_number(num_to_check):
       print(f"{num_to_check} is an Armstrong number.")
     else:
       print(f"{num_to_check} is not an Armstrong number.")
```



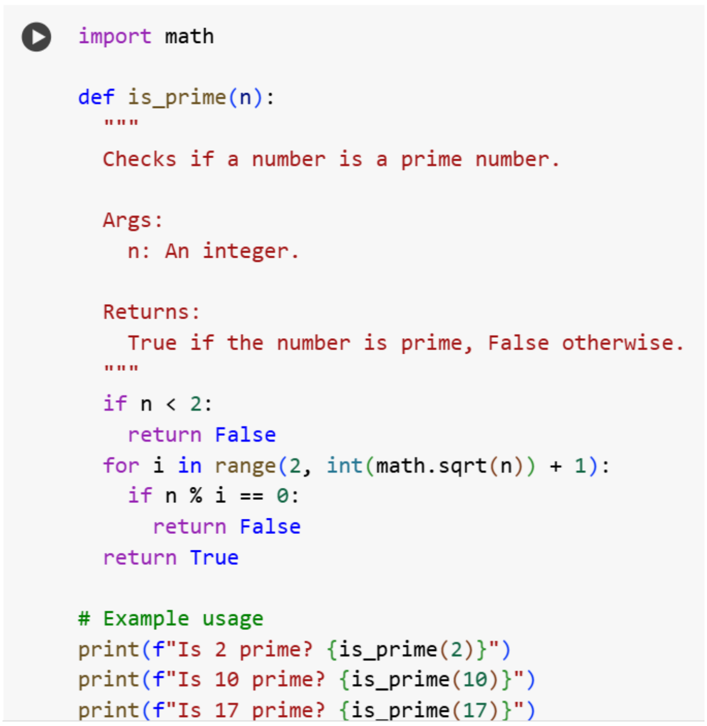
```
[30] num_to_check_2 = 123 # Example of a non-Armstrong number
     if is_armstrong_number(num_to_check_2):
       print(f"{num_to_check_2} is an Armstrong number.")
     else:
       print(f"{num_to_check_2} is not an Armstrong number.")
```

```
153 is an Armstrong number.
123 is not an Armstrong number.
```

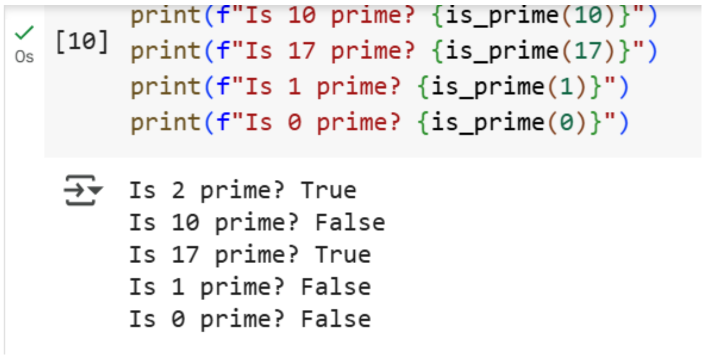
Task Description #3

● 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



```python
import math

def is_prime(n):
  """
  Checks if a number is a prime number.

  Args:
    n: An integer.

  Returns:
    True if the number is prime, False otherwise.
  """
  if n < 2:
    return False
  for i in range(2, int(math.sqrt(n)) + 1):
    if n % i == 0:
      return False
  return True

# Example usage
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is 17 prime? {is_prime(17)}")
```



```python
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is 17 prime? {is_prime(17)}")
print(f"Is 1 prime? {is_prime(1)}")
print(f"Is 0 prime? {is_prime(0)}")
```

```
Is 2 prime? True
Is 10 prime? False
Is 17 prime? True
Is 1 prime? False
Is 0 prime? False
```

EXPLANATION:

Data Analysis Key Findings

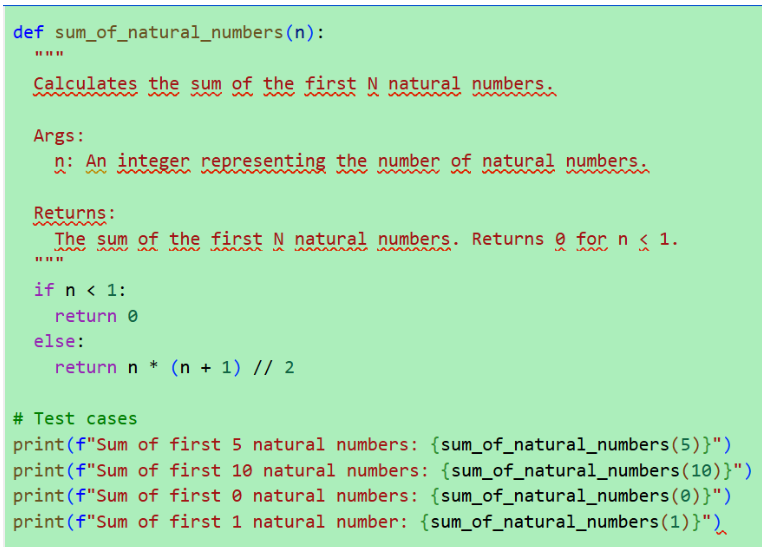
- A Python function is_prime(n) was successfully created to check if a number is prime. The function correctly handles edge cases like numbers less than 2 and utilizes an optimized approach by checking for divisors only up to the square root of n.
- A prompt was constructed to request a line-by-line explanation of the is_prime function from Gemini.
- A simulated line-by-line explanation of the is_prime function, as if provided by Gemini, was documented.

Insights or Next Steps

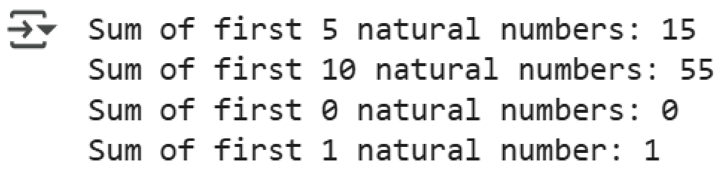
- The documented explanation can be directly used as documentation for the is_prime function, improving code readability and understanding.

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.



```python
def sum_of_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 for n < 1.
  """
  if n < 1:
    return 0
  else:
    return n * (n + 1) // 2

# Test cases
print(f"Sum of first 5 natural numbers: {sum_of_natural_numbers(5)}")
print(f"Sum of first 10 natural numbers: {sum_of_natural_numbers(10)}")
print(f"Sum of first 0 natural numbers: {sum_of_natural_numbers(0)}")
print(f"Sum of first 1 natural number: {sum_of_natural_numbers(1)}")
```

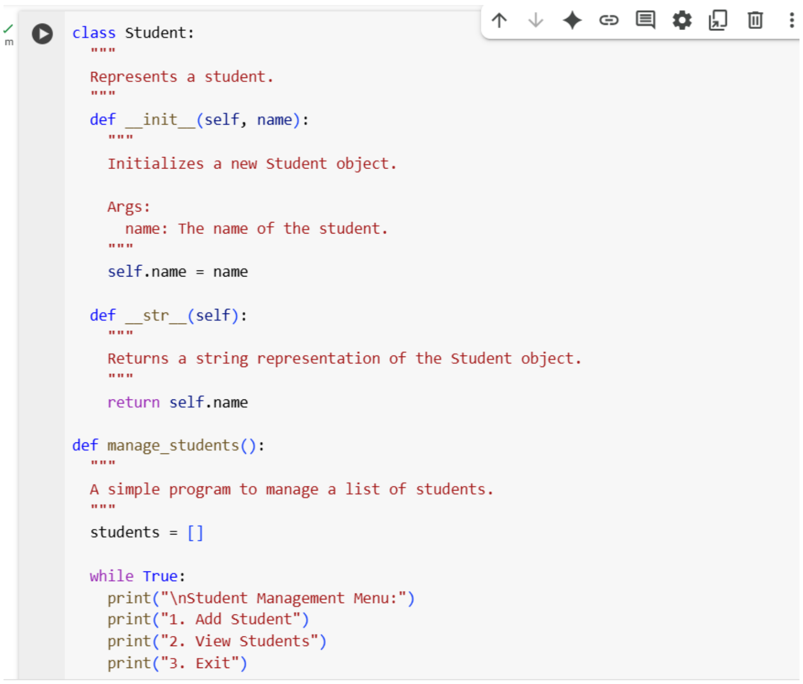


```
Sum of first 5 natural numbers: 15
Sum of first 10 natural numbers: 55
Sum of first 0 natural numbers: 0
Sum of first 1 natural number: 1
```

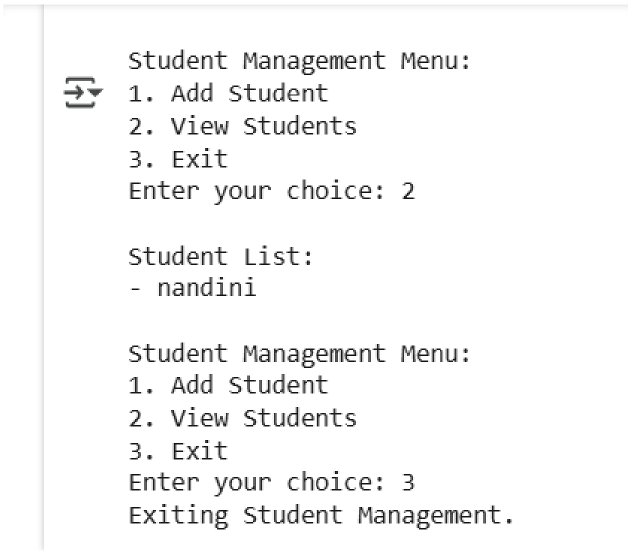
Task Description #5

● 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



```python
class Student:
  """
  Represents a student.
  """
  def __init__(self, name):
    """
    Initializes a new Student object.

    Args:
      name: The name of the student.
    """
    self.name = name

  def __str__(self):
    """
    Returns a string representation of the Student object.
    """
    return self.name

def manage_students():
  """
  A simple program to manage a list of students.
  """
  students = []

  while True:
    print("\nStudent Management Menu:")
    print("1. Add Student")
    print("2. View Students")
    print("3. Exit")
```



```
Student Management Menu:
1. Add Student
2. View Students
3. Exit
Enter your choice: 2

Student List:
- nandini

Student Management Menu:
1. Add Student
2. View Students
3. Exit
Enter your choice: 3
Exiting Student Management.
```