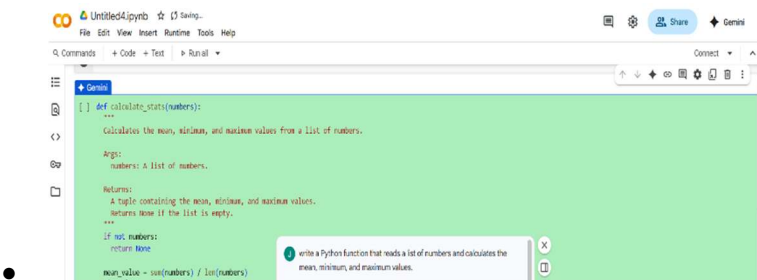
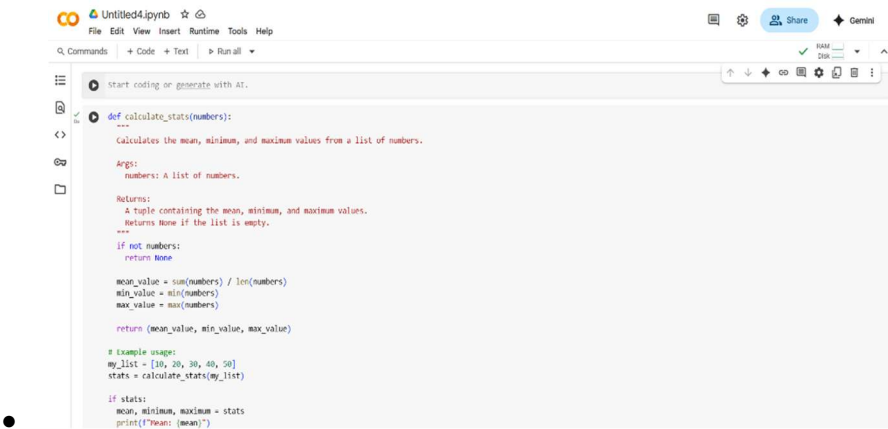


Q.No.	<div data-bbox="625 199 1036 247" data-label="Section-Header"><h1>AI ASSISTED CODING</h1></div> <div data-bbox="358 262 644 367" data-label="Text"><p>NAME:J.KEERTHI PRIYA ROLL NO:2403A510G4 ASSIGNMENT:2.1</p></div>	
1	<div data-bbox="358 415 1266 745" data-label="List-Group"><p><b>Lab Outcomes (LOs):</b> After completing this lab, students will be able to:</p><ul style="list-style-type: none"><li>● Generate Python code using Google Gemini in Google Colab.</li><li>● Analyze the effectiveness of code explanations and suggestions by Gemini.</li><li>● Set up and use Cursor AI for AI-powered coding assistance.</li><li>● Evaluate and refactor code using Cursor AI features.</li><li>● Compare AI tool behavior and code quality across different platforms.</li></ul></div> <div data-bbox="358 777 1300 940" data-label="List-Group"><p><b>Task Description #1</b></p><ul style="list-style-type: none"><li>● Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.</li></ul></div> <div data-bbox="358 945 1115 1024" data-label="List-Group"><p><b>Expected Output #1</b></p><ul style="list-style-type: none"><li>● Functional code with correct output and screenshot.</li></ul></div> <div data-bbox="410 1029 1162 1306" data-label="Image"></div> <div data-bbox="410 1312 558 1348" data-label="Section-Header"><p>● <b>CODE:</b></p></div> <div data-bbox="410 1354 1292 1780" data-label="Image"></div> <div data-bbox="410 1785 599 1820" data-label="Section-Header"><p>● <b>OUTPUT:</b></p></div>	

```
Untitled4.ipynb
File Edit View Insert Runtime Tools Help

[1] 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]
stats = calculate_stats(my_list)

if stats:
    mean, minimum, maximum = stats
    print(f"Mean: {mean}")
    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
else:
    print("The list is empty.")

Mean: 30.0
Minimum: 10
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.

## Expected Output #2

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

```
Untitled4.ipynb
File Edit View Insert Runtime Tools Help

Start coding or generate with AI.

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: An integer to check.

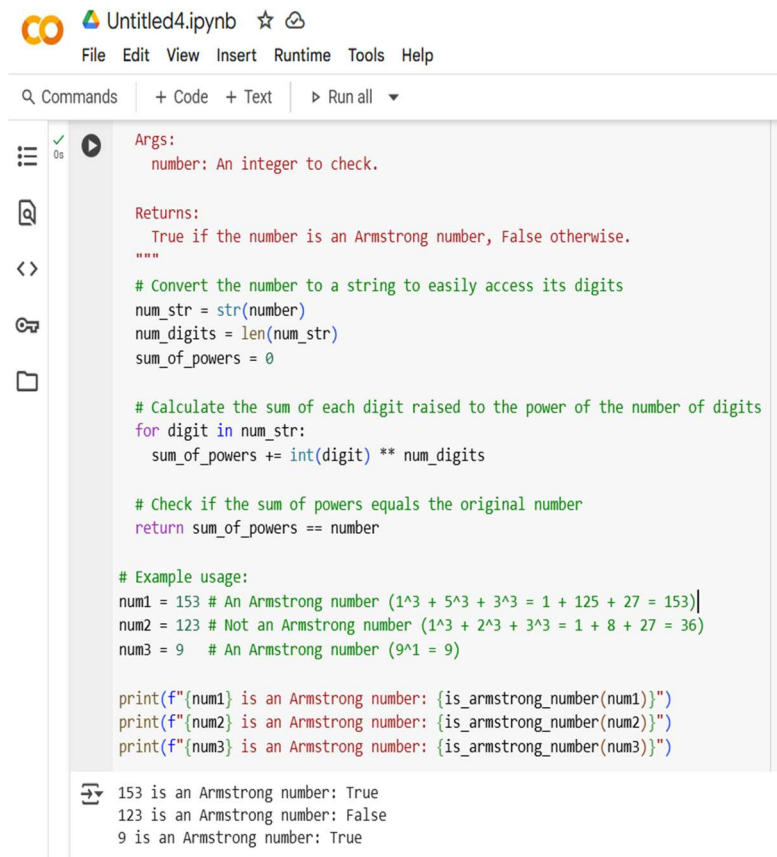
    Returns:
        True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to a string to easily access its digits
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0

    # Calculate the sum of each digit raised to the power of the number of digits
    for digit in num_str:
        sum_of_powers += int(digit) ** num_digits

    # Check if the sum of powers equals the original number
    return sum_of_powers == number

# Example usage:
print(is_armstrong_number(153))  # True
print(is_armstrong_number(10))   # False
print(is_armstrong_number(1634)) # True
print(is_armstrong_number(5454)) # False
```

- 
- **OUTPUT:**
-



The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar (Commands, + Code, + Text, Run all). The notebook contains a Python function `is_armstrong_number` that takes a number as input and returns a boolean. The function converts the number to a string to access its digits, calculates the sum of each digit raised to the power of the number of digits, and checks if this sum equals the original number. Example usage is provided with `num1 = 153`, `num2 = 123`, and `num3 = 9`. The output shows that 153 is an Armstrong number, 123 is not, and 9 is.

```
Args:
    number: An integer to check.

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

# Calculate the sum of each digit raised to the power of the number of digits
for digit in num_str:
    sum_of_powers += int(digit) ** num_digits

# Check if the sum of powers equals the original number
return sum_of_powers == number

# Example usage:
num1 = 153 # An Armstrong number (1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153)
num2 = 123 # Not an Armstrong number (1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36)
num3 = 9   # An Armstrong number (9^1 = 9)

print(f"{num1} is an Armstrong number: {is_armstrong_number(num1)}")
print(f"{num2} is an Armstrong number: {is_armstrong_number(num2)}")
print(f"{num3} is an Armstrong number: {is_armstrong_number(num3)}")
```

153 is an Armstrong number: True  
123 is an Armstrong number: False  
9 is an Armstrong number: True

- 
- **EXPLANATION:**
  - **Test with different inputs:** Try calling the function with a wider range of numbers, including larger numbers, to see how it performs.
  - **Find Armstrong numbers within a range:** Write a script or another function that iterates through a range of numbers (e.g., from 1 to 1000) and uses the `is_armstrong_number` function to identify and print all Armstrong numbers within that range.
  - **Optimize the function:** For very large numbers, converting to a string and back might not be the most efficient approach. You could explore alternative ways to extract digits and calculate the sum of
-

function that iterates through a range of numbers (e.g., from 1 to 1000) and uses the `is_armstrong_number` function to identify and print all Armstrong numbers within that range.

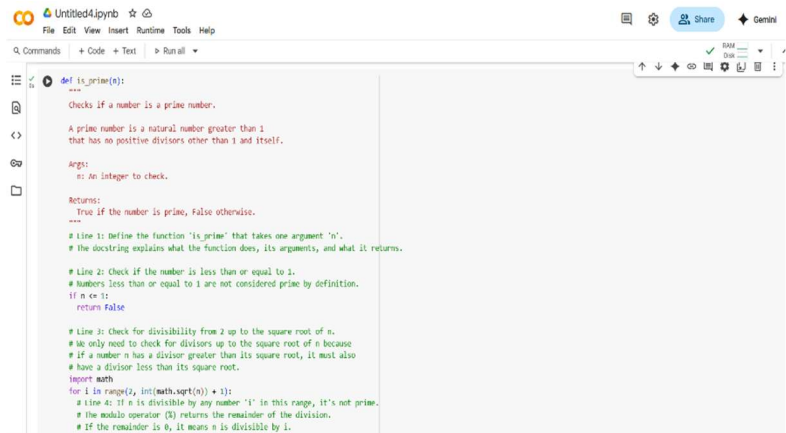
- **Optimize the function:** For very large numbers, converting to a string and back might not be the most efficient approach. You could explore alternative ways to extract digits and calculate the sum of powers using mathematical operations.
- **Explore other types of "narcissistic" numbers:** Research and implement functions to check for other types of numbers with similar properties, such as perfect digital invariants.

### 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.

### Expected Output #3

- Detailed explanation with the code snippet and Gemini's response.
- **CODE and LINE BY LINE EXPLANATION:**



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

    A prime number is a natural number greater than 1
    that has no positive divisors other than 1 and itself.

    Args:
        n: An integer to check.

    Returns:
        True if the number is prime, False otherwise.
    """
    # Line 1: Define the function 'is_prime' that takes one argument 'n'.
    # The docstring explains what the function does, its arguments, and what it returns.

    # Line 2: Check if the number is less than or equal to 1.
    # Numbers less than or equal to 1 are not considered prime by definition.
    if n <= 1:
        return False

    # Line 3: Check for divisibility from 2 up to the square root of n.
    # We only need to check for divisors up to the square root of n because
    # if a number n has a divisor greater than its square root, it must also
    # have a divisor less than its square root.
    import math
    for i in range(2, int(math.sqrt(n)) + 1):
        # Line 4: If n is divisible by any number 'i' in this range, it's not prime.
        # The modulo operator (%) returns the remainder of the division.
        # If the remainder is 0, it means n is divisible by i.
```

```
Untitled4.ipynb
File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼

# Line 3: Check for divisibility from 2 up to the square root of n.
# We only need to check for divisors up to the square root of n because
# if a number n has a divisor greater than its square root, it must also
# have a divisor less than its square root.
import math
for i in range(2, int(math.sqrt(n)) + 1):
    # Line 4: If n is divisible by any number 'i' in this range, it's not prime.
    # The modulo operator (%) returns the remainder of the division.
    # If the remainder is 0, it means n is divisible by i.
    if n % i == 0:
        return False

# Line 5: If the loop completes without finding any divisors, the number is prime.
return True

# Example usage:
num1 = 11 # Prime number
num2 = 15 # Not a prime number
num3 = 2 # Prime number
num4 = 1 # Not a prime number

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)}")

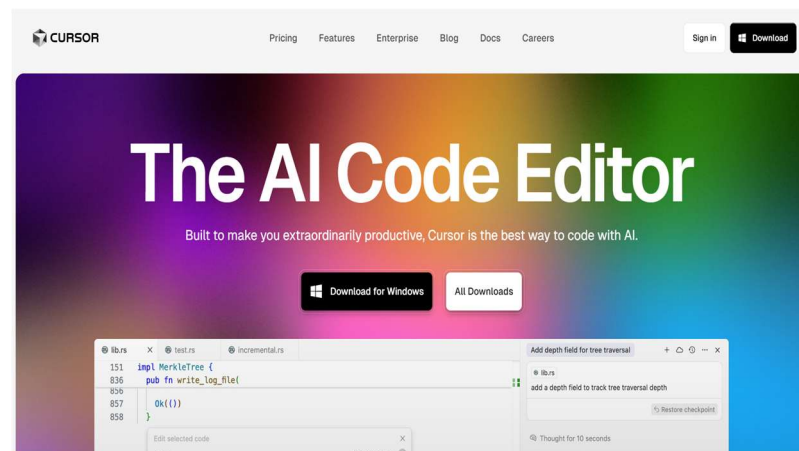
11 is a prime number: True
15 is a prime number: False
2 is a prime number: True
1 is a prime number: False
```

#### 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.

#### Expected Output #4

- Screenshots of Cursor AI setup, prompts used, and generated code with output.
- **INSTALLIZATION OF CURSOR AI:**



- **CURSOR AI's GENERATED CODE :**

```
File Edit Selection View Go Run Terminal Help • def sum_of_first_n_naturals(n): • Untitled-1 - Cursor
def sum_of_first_n_naturals(n):
    """
    Calculates the sum of the first N natural numbers.

    Args:
        n: An integer representing the number of natural numbers to sum.

    Returns:
        The sum of the first N natural numbers.
        Returns 0 if n is less than 1.
    """
    if n < 1:
        return 0
    else:
        # The sum of the first N natural numbers can be calculated using the formula: n * (n + 1) / 2
        return n * (n + 1) // 2

# Example usage:
num1 = 5
num2 = 10
num3 = 0
num4 = -3

print(f"The sum of the first {num1} natural numbers is: {sum_of_first_n_naturals(num1)}")
print(f"The sum of the first {num2} natural numbers is: {sum_of_first_n_naturals(num2)}")
print(f"The sum of the first {num3} natural numbers is: {sum_of_first_n_naturals(num3)}")
print(f"The sum of the first {num4} natural numbers is: {sum_of_first_n_naturals(num4)}")
```

- 
- **GEMINI's OUTPUT:**

```
print(f"The sum of the first {num1} natural numbers is: {sum_of_first_n_naturals(num1)}")
```

```
☞ The sum of the first 5 natural numbers is: 15
The sum of the first 10 natural numbers is: 55
The sum of the first 0 natural numbers is: 0
The sum of the first -3 natural numbers is: 0
```

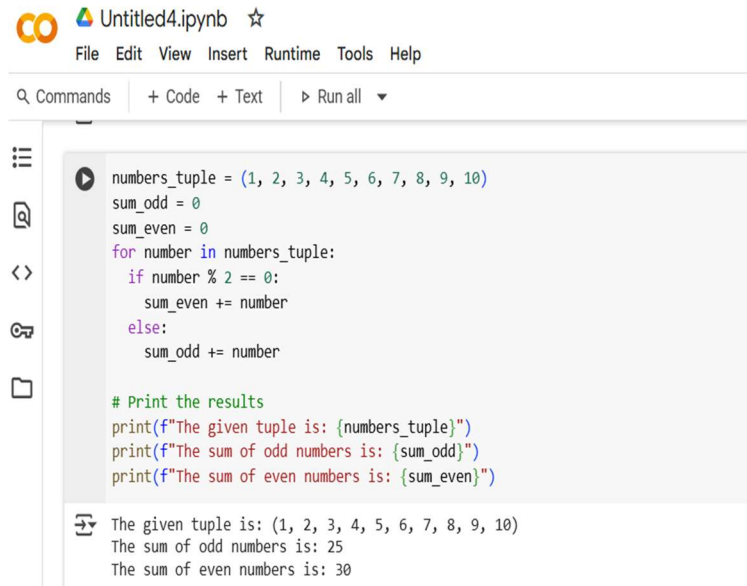
- 

### 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.

### Expected Output #5

- Student-written refactored code with explanations and output screenshots.
- **CODE WITH OUTPUT :**



The screenshot shows a Jupyter Notebook titled 'Untitled4.ipynb'. The code defines a tuple 'numbers\_tuple' with values (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). It initializes 'sum\_odd' and 'sum\_even' to 0. A for loop iterates through the tuple, adding numbers to 'sum\_even' if they are even (number % 2 == 0) and to 'sum\_odd' if they are odd. Finally, it prints the tuple and the sums. The output shows the tuple and the sums: 25 for odd numbers and 30 for even numbers.

```
numbers_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
sum_odd = 0
sum_even = 0
for number in numbers_tuple:
    if number % 2 == 0:
        sum_even += number
    else:
        sum_odd += number

# Print the results
print(f"The given tuple is: {numbers_tuple}")
print(f"The sum of odd numbers is: {sum_odd}")
print(f"The sum of even numbers is: {sum_even}")
```

The given tuple is: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
The sum of odd numbers is: 25  
The sum of even numbers is: 30

**Note:**

- Students must submit a single Word document including:
  - Prompts used for AI tools
  - Copilot/Gemini/Cursor outputs
  - Code explanations
  - Screenshots of outputs and environments

**Evaluation Criteria:**

Criteria	Max Marks
Successful Use of Gemini in Colab (Task#1 & #2)	1.0
Code Explanation Accuracy (Gemini) (Task#3)	0.5
Cursor AI Setup and Usage (Task#4)	0.5
Refactoring and Improvement Analysis (Task#5)	0.5
<b>Total</b>	<b>2.5 Marks</b>