
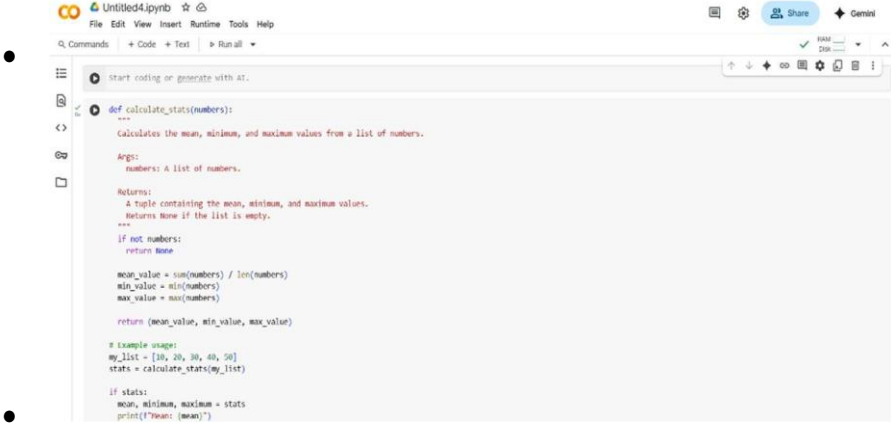


<p><b>Q.No.</b></p>	<p align="center"><b>AI ASSISTED CODING</b></p> <p><b>NAME:B.SRISHANTH</b>  <b>ROLL NO:2403A510G3</b>  <b>ASSIGNMENT:2.1</b></p>	
<p align="center">1</p>	<p><b>Lab Outcomes (LOs):</b>  After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> <li>● 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> <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> <p><b>Expected Output #1</b></p> <div>  </div> <div>  </div> <ul style="list-style-type: none"> <li>● <b>OUTPUT:</b></li> </ul>	

```
Untitled4.ipynb
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
[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

```
Untitled4.ipynb
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
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.

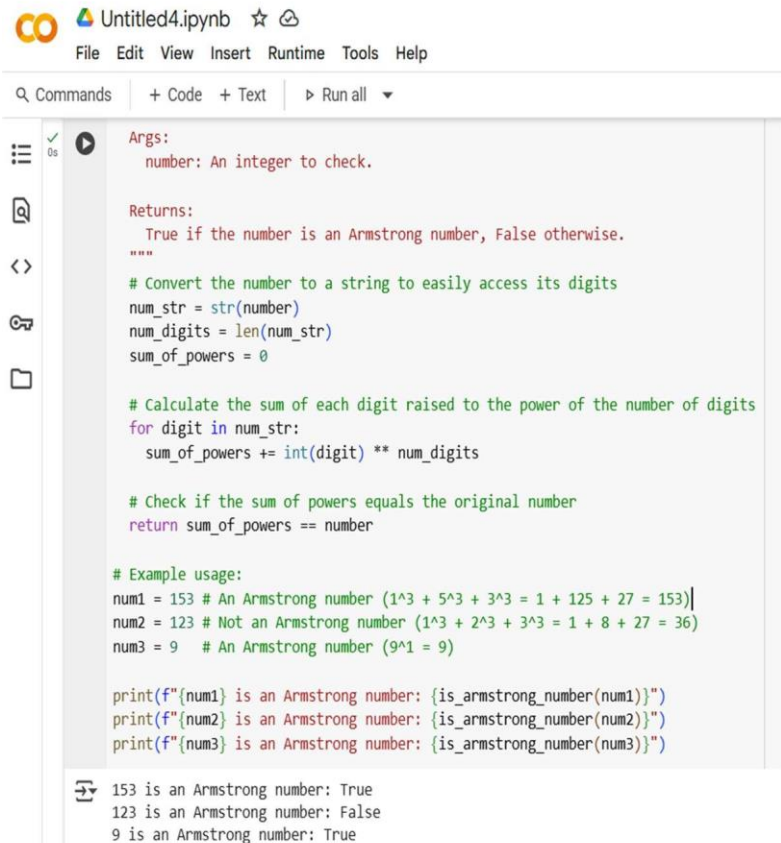
    Returns:
        True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to a string to easily access its digits
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0

    # 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:
```

- **OUTPUT:**



The screenshot shows a Jupyter Notebook interface with the title 'Untitled4.ipynb'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu bar, there are tabs for 'Commands', '+ Code', '+ Text', and 'Run all'. The notebook contains a Python function `is_armstrong_number` with the following code:

```
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)}")
```

The output of the notebook shows the results of the function calls:

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

- 
- 
- **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.

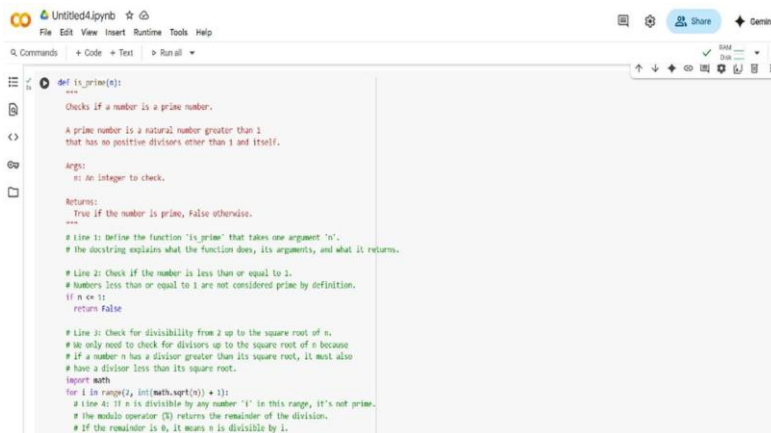
- **Optimize the function:** For very large numbers, converting to a string and back might not be the most efficient approach. You could explore alternative ways to extract digits and calculate the sum of powers using mathematical operations.
- **Explore other types of "narcissistic" numbers:** Research 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



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

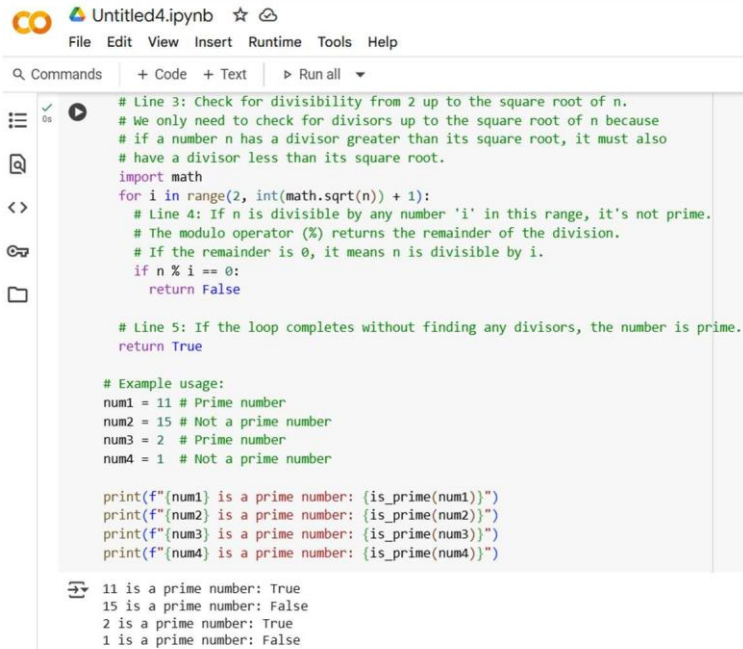
    A prime number is a natural number greater than 1
    that has no positive divisors other than 1 and itself.

    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.
```



```

# 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

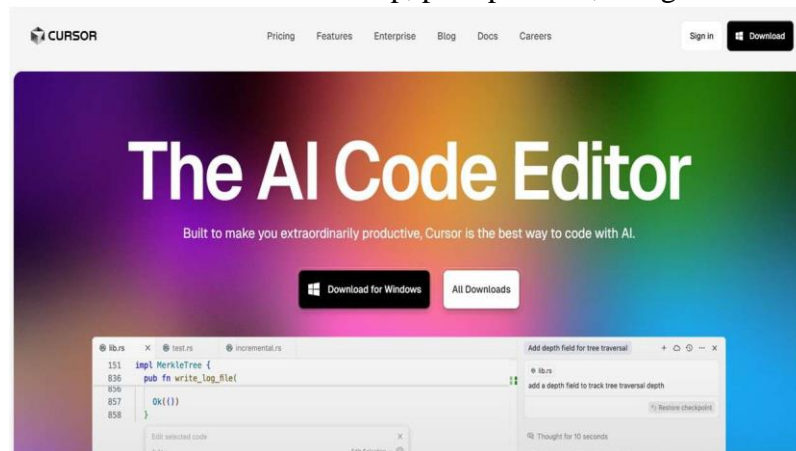
```

#### 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



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



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 odd numbers to 'sum\_odd' and even numbers to 'sum\_even'. Finally, it prints the tuple and the two sums.

```

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 output of the code is displayed below the cell:

```

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>