SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name STUDENT DETAILS:	Venkataramana Veeramsetty. **Peddapelli Anusha 2403A51103 Batch-06.**		
Instructor(s) Name	Dr. V. Venkataramana (Co-Ordinator)		
	Dr. T. Sampath Kumar		
	Dr. Pramoda Patro		
	Dr. Brij Kishor Tiwari		
	Dr.J.Ravichander		
	Dr. Mohammand Ali Shaik		
	Dr. Anirodh Kumar		
	Mr. S.Naresh Kumar		
	Dr. RAJESH VELPULA		
	Mr. Kundhan Kumar		
	Ms. Ch.Rajitha		
	Mr. M Prakash		
	Mr. B.Raju		
	Intern 1 (Dharma teja)		
	Intern 2 (Sai Prasad)		
	Intern 3 (Sowmya)		
	NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week1 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	24CSBTB01 To 24CSBTB39

Assignment Number: 2.1(Present assignment number)/**24**(Total number of assignments)

Q.No.	Question	Expected Time to complete
1	Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and	Week1 - Monday

Cursor AI

Lab Objectives:

- To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab.
- To understand and use Cursor AI for code generation, explanation, and refactoring.
- To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI.
- To perform code optimization and documentation using AI tools.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Generate Python code using Google Gemini in Google Colab.
- Analyze the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using Cursor AI features.
- Compare AI tool behavior and code quality across different platforms.

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.

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.

```
Start coding or generate with AI.
```

```python
def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum values from a list of numbers.

    Args:
      numbers: A list of numbers (integers or floats).

    Returns:
      A dictionary containing the mean, minimum, and maximum values.
      Returns None if the input list is empty.
    """
    if not numbers:
      return None

    mean_value = sum(numbers) / len(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

    return {
        "mean": mean_value,
        "minimum": min_value,
        "maximum": max_value
    }
```

```python
        mean_value = sum(numbers) / len(numbers)
        min_value = min(numbers)
        max_value = max(numbers)

        return {
            "mean": mean_value,
            "minimum": min_value,
            "maximum": max_value
        }

# Example usage:
my_list = [10, 20, 30, 40, 50]
result = analyze_numbers(my_list)

if result:
    print(f"Analysis of the list {my_list}:")
    print(f"Mean: {result['mean']}")
    print(f"Minimum: {result['minimum']}")
    print(f"Maximum: {result['maximum']}")
else:
    print("The list is empty.")
```

```
Analysis of the list [10, 20, 30, 40, 50]:
Mean: 30.0
Minimum: 10
Maximum: 50
```

Double-click (or enter) to edit

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.

Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

▶ Start coding or generate with AI.

Double-click (or enter) to edit

```python
[2] 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.

    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)
    armstrong_sum = 0

    # Calculate the sum of each digit raised to the power of the number of digits
    for digit in num_str:
```

```
        # Convert the number to a string to easily access its digits
        num_str = str(number)
        num_digits = len(num_str)
        armstrong_sum = 0

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

        # Check if the calculated sum is equal to the original number
        return armstrong_sum == number

    # Example usage:
    num_to_check = 153
    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.")

    num_to_check = 123
    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.")
```

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

```
def is_armstrong(number):
```
🔵 Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

| Ask or edit in context | GPT-4.1 ⌄ |

Accept   Close   ↻   ⌄

```
    """
    Checks whether a number is an Armstrong number.

    Steps:
    1. Convert the number to a string to count its digits.
    2. For each digit, raise it to the power of the number of digits.
    3. Sum these powered digits.
    4. If the sum equals the original number, it is an Armstrong number.

    Prompt:
    Enter a number to check if it is an Armstrong number.
```
r\Desktop\B.Tech\First Year(1)\Python Programs\partterns.py
```
    Output:
    Prints whether the number is an Armstrong number or not.
    """
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == number
```
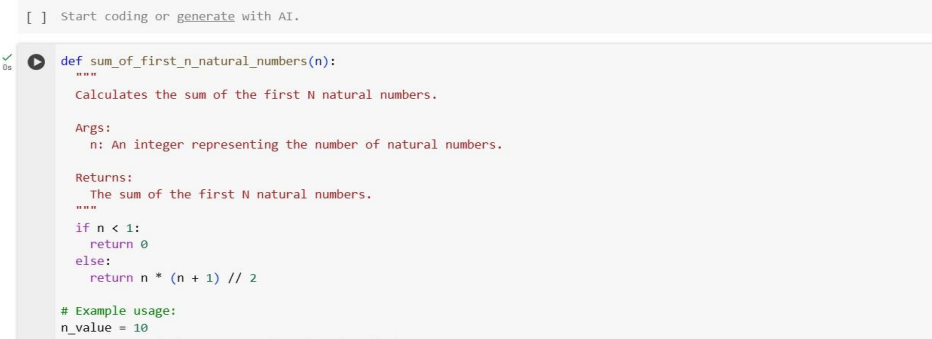
Feature	Gemini Code	Copilot Code
Function Name	`is_armstrong_number(number)`	`is_armstrong(number)`
Logic Style	String conversion + `for` loop	Arithmetic with `%` and `while` loop
Docstring	🟩 Detailed with Args & Returns	∣ Minimal or none
Readability	High – commented and well-structured	Moderate – more compact, sometimes no comments
Test Cases	🟩 Shown (153, 123)	🟩 Usually includes 153; others need prompting
Output Example	`153 is an Armstrong number.123 is not an Armstrong number.`	Same output

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.

● 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.

```
Start coding or generate with AI.
```

```
[ ] Start coding or generate with AI.
```

Steps to check for an Armstrong number:

1. **Define the function `is_armstrong_number(number)`:** This function takes an integer `number` as input.
2. **Convert the number to a string:** This allows easy iteration through the digits of the number.
3. **Get the number of digits:** Determine the length of the string representation of the number.
4. **Initialize `armstrong_sum` to 0:** This variable will store the sum of the digits raised to the power of the number of digits.
5. **Iterate through each digit:**
   ○ Convert the digit back to an integer.
   ○ Raise the integer digit to the power of the total number of digits.
   ○ Add the result to `armstrong_sum`.
6. **Compare the sum with the original number:** If `armstrong_sum` is equal to the original `number`, it is an Armstrong number.
7. **Return True or False:** Based on the comparison result.

```python
def is_palindrome(s):
    """
    Checks if a string is a palindrome.

    A palindrome is a string that reads the same forwards and backwards.

    Args:
      s: The input string.

    Returns:
      True if the string is a palindrome, False otherwise.
    """
    # Remove spaces and convert to lowercase for case-insensitive check
    s = s.replace(" ", "").lower()
    # Compare the string with its reverse
    return s == s[::-1]

# Example usage:
text1 = "racecar"
text2 = "hello world"
text3 = "Madam"

print(f"'{text1}' is a palindrome: {is_palindrome(text1)}")
print(f"'{text2}' is a palindrome: {is_palindrome(text2)}")
print(f"'{text3}' is a palindrome: {is_palindrome(text3)}")
```

```
'racecar' is a palindrome: True
'hello world' is a palindrome: False
'Madam' is a palindrome: 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.

Expected Output #4

- Screenshots of Cursor AI setup, prompts used, and generated code with output.

● 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.

```
[ ]  Start coding or generate with AI.
```

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

# Example usage:
n_value = 10
```

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

# Example usage:
n_value = 10
sum_n = sum_of_first_n_natural_numbers(n_value)
print(f"The sum of the first {n_value} natural numbers is: {sum_n}")

n_value = 5
sum_n = sum_of_first_n_natural_numbers(n_value)
print(f"The sum of the first {n_value} natural numbers is: {sum_n}")
```

```
The sum of the first 10 natural numbers is: 55
The sum of the first 5 natural numbers is: 15
```

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

Python program to calculate the sum of odd numbers and even numbers in a given tuple. ● Refactor the code to improve logic and reada

```python
def sum_odd_even(numbers_tuple):
    """
    Calculates the sum of odd and even numbers in a tuple.

    Args:
      numbers_tuple: A tuple of numbers (integers or floats).

    Returns:
      A dictionary containing the sum of odd numbers and the sum of even numbers.
    """
    sum_even = 0
    sum_odd = 0

    for number in numbers_tuple:
      # Check if the number is an integer before performing modulo operation
      if isinstance(number, int):
        if number % 2 == 0:
          sum_even += number
        else:
          sum_odd += number
      # You might want to handle non-integer types differently,
      # for now, we'll skip them for the odd/even check.

    return {"sum_even": sum_even, "sum_odd": sum_odd}
```

```python
        sum_odd += number
      # You might want to handle non-integer types differently,
      # for now, we'll skip them for the odd/even check.

    return {"sum_even": sum_even, "sum_odd": sum_odd}

# Example usage:
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
result = sum_odd_even(my_tuple)

print(f"Original tuple: {my_tuple}")
print(f"Sum of even numbers: {result['sum_even']}")
print(f"Sum of odd numbers: {result['sum_odd']}")

my_tuple_2 = (11, 22, 33, 44, 55, 66)
result_2 = sum_odd_even(my_tuple_2)

print(f"\nOriginal tuple: {my_tuple_2}")
print(f"Sum of even numbers: {result_2['sum_even']}")
print(f"Sum of odd numbers: {result_2['sum_odd']}")
```

```
Original tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum of even numbers: 30
Sum of odd numbers: 25

Original tuple: (11, 22, 33, 44, 55, 66)
Sum of even numbers: 132
Sum of odd numbers: 99
```

Note:
- Students must submit a single Word document including:
  - Prompts used for AI tools

- o Copilot/Gemini/Cursor outputs
- o Code explanations
- o 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
Total	**2.5 Marks**