SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName: B. Tech		**Assignment Type: Lab**	**AcademicYear:** 2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
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)	
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week1 - Thursday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	24CSBTB01 To 24CSBTB39
AssignmentNumber: **2.4**(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*ExpectedTime to complete*
1	Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI **Lab Objectives:**	Week1 - Thursday

- 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
• Open Google Colab and use Google Gemini to generate Python code that performs sorting of a list using both the bubble sort algorithm and Python's built-in sort() function. Compare the two implementations.
Expected Output #1
• Two sorting implementations: Bubble sort (manual logic) and Built-in sort()
Prompt #1:
1.Creates a list of random numbers.
2. Implements bubble sort to sort the list.
3. Uses Python's built-in sort() function to sort the same list.
4. Compares the implementations of both methods and prints the results.
Observations:
1. Both methods sorted the list correctly.
2. Bubble sort took more time than the built-in sort.
3. This happens because bubble sort is slower than Python's sort .
4. Bubble sort is good for learning, but built-in sort is better for real use

CODE:.

```python
import random
import time
random_list = [random.randint(0, 1000) for _ in range(100)]
def bubble_sort(arr):
    """
    Sorts a list using the bubble sort algorithm.
    """
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
start_time_bubble = time.time()
bubble_sorted_list = bubble_sort(random_list.copy())
end_time_bubble = time.time()
start_time_built_in = time.time()
built_in_sorted_list = sorted(random_list.copy())
end_time_built_in = time.time()
print("Original list (first 10 elements):", random_list[:10])
print("Bubble sorted list (first 10 elements):", bubble_sorted_list[:10])
print("Built-in sorted list (first 10 elements):", built_in_sorted_list[:10])
print("\nComparison:")
print(f"Bubble sort took: {end_time_bubble - start_time_bubble:.6f} seconds")
print(f"Built-in sort() took: {end_time_built_in - start_time_built_in:.6f} seconds")
print("\nVerification:")
print("Are both lists sorted correctly?", bubble_sorted_list == built_in_sorted_list)
```

```
Original list (first 10 elements): [660, 324, 636, 655, 66, 591, 709, 686, 772, 978]
Bubble sorted list (first 10 elements): [4, 66, 80, 83, 84, 100, 126, 128, 129, 148]
Built-in sorted list (first 10 elements): [4, 66, 80, 83, 84, 100, 126, 128, 129, 148]

Comparison:
Bubble sort took: 0.000454 seconds
Built-in sort() took: 0.000058 seconds

Verification:
Are both lists sorted correctly? True
```

Task Description #2

• In Colab, use Google Gemini to generate a Python function that takes a string and returns: The number of vowels, The number of consonants, The number of digits in the string

Expected Output #2-

• Complete function that Iterates through characters of a string and Counts vowels, consonants, and digits

Prompt #2

1. Defines a function that takes a string as input.
2. Counts the number of vowels in the string.
3. Counts the number of consonants in the string.
4. Counts the number of digits in the string.
5. Returns these counts.

Observations:

1. The function correctly counts vowels, consonants, and digits.
2. It works for both uppercase and lowercase letters.
3. Non-alphabetic, non-digit characters are ignored in the counts.
4. Useful for basic text analysis tasks.

CODE:

```python
def count_chars(input_string: str) -> tuple[int, int, int]:
    """
    Counts the number of vowels, consonants, and digits in a string.
    Args:
        input_string: The string to analyze.
    Returns:
        A tuple containing:
        (vowel_count, consonant_count, digit_count)
    """
    vowel_count = 0
    consonant_count = 0
    digit_count = 0
    vowels = "aeiouAEIOU"
    for char in input_string:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif char.isdigit():
            digit_count += 1
    return vowel_count, consonant_count, digit_count
text = input("Enter a string: ")
v, c, d = count_chars(text)
print(f"Vowels: {v}")
print(f"Consonants: {c}")
print(f"Digits: {d}")
```

```
Enter a string: Sru123
Vowels: 1
Consonants: 2
Digits: 3
```

Task Description #3

• Install and set up Cursor AI. Use it to generate a Python program that performs file handling:
Create a text file
Write sample text
Read and display the content

Expected Output #3

• Functional code that creates a .txt file, writes content to it, and reads it back.
• Screenshot of Cursor AI interface showing: Prompt used,
Generated code, Output of file operations

Prompt #3:

Write a Python program that:
1. Creates a text file named "sample.txt".
2. Writes some sample text into the file.
3. Reads the file content.
4. Displays the content on the screen.
Use proper file handling methods.

Observations:

1. The program **creates** a new text file.
2. It **writes** sample text into the file using write mode (`"w"`).
3. It **reads** the content back using read mode (`"r"`).
4. The file content is **displayed** on the screen.

CODE:

```python
# Create a text file and write to it
file_name = "sample.txt"
with open(file_name, "w") as file:
    file.write("This is some sample text.\n")
    file.write("This is the second line.")

# Read and display the content of the file
with open(file_name, "r") as file:
    content = file.read()
    print("File content:")
    print(content)
```

```
File content:
This is some sample text.
This is the second line.
```

Task Description #4

• Ask Google Gemini to generate a Python program that implements a simple calculator using functions (add, subtract, multiply, divide). Then, ask Gemini to explain how the code works.

Expected Output #4

• Complete calculator code with user input and operation selection.

• Line-by-line explanation or markdown-style explanation provided by Gemini.

• Screenshot of both the code and explanation in Colab.

Prompt #4:

1.Implements a simple calculator using functions for add, subtract, multiply, and divide.
2. Takes two numbers from the user.
3. Asks the user which operation to perform.
4. Displays the result.

Observations:

1. The program uses **functions** to perform each operation (add, subtract, multiply, divide).
2. It **takes two numbers** from the user as input.
3. The user can **choose an operation** by entering a choice.
4. The selected function is **called** to perform the calculation.
5. The **result is displayed** to the user.

CODE:

```
def add(x, y):
  """Adds two numbers."""
  return x + y

def subtract(x, y):
  """Subtracts the second number from the first."""
  return x - y

def multiply(x, y):
  """Multiplies two numbers."""
  return x * y

def divide(x, y):
  """Divides the first number by the second."""
  if y == 0:
    return "Error: Division by zero"
  return x / y

print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")

while True:
  choice = input("Enter choice(1/2/3/4): ")

  if choice in ['1', '2', '3', '4']:
    try:
      num1 = float(input("Enter first number: "))
      num2 = float(input("Enter second number: "))
    except ValueError:
      print("Invalid input. Please enter numbers.")
      continue

    if choice == '1':
      print(num1, "+", num2, "=", add(num1, num2))
    elif choice == '2':
      print(num1, "-", num2, "=", subtract(num1, num2))
    elif choice == '3':
      print(num1, "*", num2, "=", multiply(num1, num2))
    elif choice == '4':
      print(num1, "/", num2, "=", divide(num1, num2))
    break
  else:
    print("Invalid Input")
```

```
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4): 1
Enter first number: 23
Enter second number: 12
23.0 + 12.0 = 35.0
```

Task Description #5

• Use Cursor AI to create a Python program that checks if a given year is a leap year or not. Try different prompt styles and see how Cursor modifies its code suggestions.

Expected Output #5

• A functional program to check leap year with sample input/output

• At least two versions of the code (from different prompts)

• A short comparison of which version is better and why

Prompt #5:

1.Takes a year as input from the user.
2. Checks if the year is a leap year or not.
3. Displays the result.

Observations:

1. The program **takes a year** as input from the user.
2. It uses the **leap year rule**
3. The program **checks the condition** using if statements.
4. The logic works for both **past and future years**.

Code:

```
def is_leap(year):
  """Checks if a given year is a leap year."""
  return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

# Get input from the user
try:
  year = int(input("Enter a year: "))
  if is_leap(year):
    print(f"{year} is a leap year")
  else:
    print(f"{year} is not a leap year")
except ValueError:
  print("Invalid input. Please enter an integer.")
```

```
Enter a year: 2024
2024 is a leap year
```

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	Max Marks
Two sorting implementations: Bubble sort (manual logic) and Built-in sort() (Task#1)	0.5
Counts vowels, consonants, and digits(Task#2)	0.5
Functional code that creates a .txt file, writes content to it, and reads it back- Use cursor (Task#3)	0.5
Complete calculator code with user input and operation selection. (Task#4)	0.5
A functional program to check leap year with sample input/output-use Cursor (Task#5)	0.5
Total	**2.5 Marks**