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
• # Write Python code that sorts a list in two ways:
 # 1. Using bubble sort algorithm (manual implementation)
 # 2. Using Python's built-in sort() method
 # Print both sorted lists for comparison

```
def bubble_sort(arr):
    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

# Original list
my_list = [64, 34, 25, 12, 22, 11, 90]

# Sort using bubble sort
bubble_sorted_list = bubble_sort(my_list.copy()) # Use a copy to avoid modifying the original list
print("Bubble Sorted List:", bubble_sorted_list)

# Sort using Python's built-in sort() method
my_list.sort()
print("Built-in Sorted List:", my_list)
```

Expected Output #1
• Two sorting implementations: Bubble sort (manual logic) and Built-in sort()

```
Bubble Sorted List: [11, 12, 22, 25, 34, 64, 90]
Built-in Sorted List: [11, 12, 22, 25, 34, 64, 90]
```

This notebook sorts a list of numbers in two ways:

1. **Bubble Sort (Manual Implementation)**: The `bubble_sort` function implements the bubble sort algorithm. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted. A copy of the original list is used so that the original list remains unchanged for the second sorting method.
2. **Python's Built-in `sort()` Method**: The code then uses Python's built-in `sort()` method directly on the original list. This method sorts the list in-place, meaning it modifies the original list directly.

Finally, both the bubble-sorted list and the list sorted by the built-in method are printed to show the results.

Task Description #2
• # Write a Python function that takes a string and returns:
 # - Number of vowels
 # - Number of consonants
 # - Number of digits
 # Test the function with an example string and print the results

```
def count_chars(input_string):
    vowels = "aeiouAEIOU"
    consonants = "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
    digits = "0123456789"

    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in input_string:
        if char in vowels:
            vowel_count += 1
        elif char in consonants:
            consonant_count += 1
        elif char in digits:
            digit_count += 1

    return vowel_count, consonant_count, digit_count

# Test the function with an example string
example_string = "Hello World 123!"
vowels, consonants, digits = count_chars(example_string)

print(f"Input string: '{example_string}'")
print(f"Number of vowels: {vowels}")
print(f"Number of consonants: {consonants}")
print(f"Number of digits: {digits}")
```

Expected Output #2-
• Complete function that Iterates through characters of a string and Counts vowels, consonants, and digits

```
Input string: 'Hello World 123!'
Number of vowels: 3
Number of consonants: 7
Number of digits: 3
```

1. `def count_chars(input_string):` : This line defines the function `count_chars` that accepts one argument, `input_string`.
2. `vowels = "aeiouAEIOU"`, `consonants = "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"`, `digits = "0123456789"` : These lines initialize strings containing all the characters that will be considered vowels, consonants, and digits.
3. `vowel_count = 0`, `consonant_count = 0`, `digit_count = 0` : These lines initialize counters for each category to zero.
4. `for char in input_string:` : This loop iterates through each character in the input string.
5. `if char in vowels:` : This condition checks if the current character is present in the `vowels` string. If it is, the `vowel_count` is incremented.
6. `elif char in consonants:` : If the character is not a vowel, this condition checks if it's in the `consonants` string. If it is, the `consonant_count` is incremented.
7. `elif char in digits:` : If the character is neither a vowel nor a consonant, this condition checks if it's in the `digits` string. If it is, the `digit_count` is incremented.
8. `return vowel_count, consonant_count, digit_count` : The function returns the final counts of vowels, consonants, and digits as a tuple.

Task Description #3

- **# Write Python code to:**

 # 1. Create a text file
 # 2. Write some sample text into it
 # 3. Read the text from the file and display it

```python
# 1. Create a text file and write to it
file_name = "sample.txt"
sample_text = "This is some sample text that will be written to the file."

with open(file_name, "w") as file:
    file.write(sample_text)

print(f"Successfully created and wrote to '{file_name}'")

# 2. Read the text from the file and display it
with open(file_name, "r") as file:
    read_text = file.read()

print(f"\nContent of '{file_name}':")
print(read_text)
```

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

```
Successfully created and wrote to 'sample.txt'

Content of 'sample.txt':
This is some sample text that will be written to the file.
```

1. `file_name = "sample.txt"`: This line assigns the string "sample.txt" to the variable `file_name`. This will be the name of the text file we create.
2. `sample_text = "This is some sample text that will be written to the file."`: This line assigns a sample string to the variable `sample_text`. This is the content we will write into the file.
3. `with open(file_name, "w") as file:`: This is a standard way to open a file in Python.
    - `open(file_name, "w")` opens the file named "sample.txt" in write mode (`"w"`). If the file doesn't exist, it will be created. If it does exist, its contents will be overwritten.
    - `with ... as file:` is a context manager that ensures the file is automatically closed even if errors occur. The opened file object is assigned to the variable `file`.
4. `file.write(sample_text)`: This line writes the content of the `sample_text` variable into the opened file.
5. `print(f"Successfully created and wrote to '{file_name}'")`: This line prints a confirmation message to the console indicating that the file was created and written to.
6. `with open(file_name, "r") as file:`: This opens the same file ("sample.txt") again, but this time in read mode (`"r"`).
7. `read_text = file.read()`: This line reads the entire content of the file and stores it in the `read_text` variable.
8. `print(f"\nContent of '{file_name}':")`: This prints a header before displaying the file content.
9. `print(read_text)`: This line prints the content that was read from the file to the console.

Task Description #4

• # Write a Python program for a simple calculator using functions:
- add, subtract, multiply, divide
Ask the user to select an operation and enter two numbers
Perform the calculation and print the result.

```
while True:
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    print("5. Exit")

    choice = input("Enter choice(1/2/3/4/5): ")

    if choice == '5':
        print("Exiting calculator.")
        break

    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':
            result = divide(num1, num2)
            print(num1, "/", num2, "=", result)
    else:
        print("Invalid Input")
```

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.

```
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Enter choice(1/2/3/4/5): 1
Enter first number: 1
Enter second number: 2
1.0 + 2.0 = 3.0
```

1. `while True:` : This creates an infinite loop, allowing the calculator to run until explicitly told to stop.
2. `print(...)` : These lines display the menu of available operations to the user.
3. `choice = input("Enter choice(1/2/3/4/5): ")` : This line prompts the user to enter their choice of operation and stores it in the `choice` variable.
4. `if choice == '5': ... break` : This checks if the user entered '5'. If so, it prints an exit message and the `break` statement exits the `while` loop, ending the program.
5. `if choice in ('1', '2', '3', '4'):` : This checks if the user's choice is one of the valid operation options (1, 2, 3, or 4).
6. `try:... except ValueError:...` : This is a `try-except` block for error handling. It attempts to convert the user's input for the two numbers into floating-point numbers using `float()` .
    - If the conversion is successful, the code proceeds.
    - If a `ValueError` occurs (e.g., the user enters text instead of numbers), the `except` block is executed, printing an "Invalid input" message, and `continue` skips the rest of the current loop iteration, prompting the user for input again.
7. `if choice == '1': ... elif choice == '2': ...` **(and so on)**: These `if-elif` statements check the user's valid choice and call the corresponding function (`add` , `subtract` , `multiply` , or `divide`) with the entered numbers (`num1` and `num2`).
8. `print(num1, "+", num2, "=", add(num1, num2))` : This line prints the calculation and its result for the addition operation. Similar lines exist for the other operations. For division, it prints the `result` variable, which might contain the division result or the "Error! Division by zero." message.
9. `else: print("Invalid Input")` : If the user's initial choice is not one of the valid options (1-5), this `else` block is executed, printing "Invalid Input".

Task Description #5

• # Write a Python program that checks if a given year is a leap year.

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

# Example usage:
year_to_check = 2024
if is_leap_year(year_to_check):
  print(f"{year_to_check} is a leap year.")
else:
  print(f"{year_to_check} is not a leap year.")

year_to_check = 1900
if is_leap_year(year_to_check):
  print(f"{year_to_check} is a leap year.")
else:
  print(f"{year_to_check} is not a leap year.")

year_to_check = 2000
if is_leap_year(year_to_check):
  print(f"{year_to_check} is a leap year.")
else:
  print(f"{year_to_check} is not a leap year.")
```

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

```
2024 is a leap year.
1900 is not a leap year.
2000 is a leap year.
```

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

- `def is_leap_year(year):` : This line defines a function named `is_leap_year` that accepts one argument, `year` .
- `"""Checks if a given year is a leap year."""` : This is a docstring, which explains what the function does.
- `return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)` : This is the core logic. It checks the leap year conditions:
    - `year % 4 == 0` : Checks if the year is divisible by 4.
    - `year % 100 != 0` : Checks if the year is NOT divisible by 100.
    - `year % 400 == 0` : Checks if the year is divisible by 400.
    - The `and` operator combines the first two conditions: a year is a leap year if it's divisible by 4 AND not by 100.
    - The `or` operator combines the result of the first part with the third condition: a year is also a leap year if it's divisible by 400 (this handles the exception to the divisible by 100 rule).

```python
# Example usage:
year_to_check = 2024
if is_leap_year(year_to_check):
  print(f"{year_to_check} is a leap year.")
else:
  print(f"{year_to_check} is not a leap year.")
```

- `# Example usage:` : This is a comment indicating the following lines are examples.
- `year_to_check = 2024` : This line assigns the integer 2024 to the variable `year_to_check`.
- `if is_leap_year(year_to_check):` : This line calls the `is_leap_year` function with `year_to_check` (2024) and checks if the returned value is `True`.
- `print(f"{year_to_check} is a leap year.")` : If the `if` condition is `True` (2024 is a leap year), this line prints a formatted string stating that 2024 is a leap year.
- `else:` : This introduces the alternative block of code to be executed if the `if` condition i `False`.
- `print(f"{year_to_check} is not a leap year.")` : If the `if` condition is `False`, this line prints a formatted string stating that 2024 is not 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**