| | |
|---|---|
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** |

| | | |
|---|---|---|
| **ProgramName:**B. Tech | **Assignment Type: Lab** | **AcademicYear:**2025-2026 |

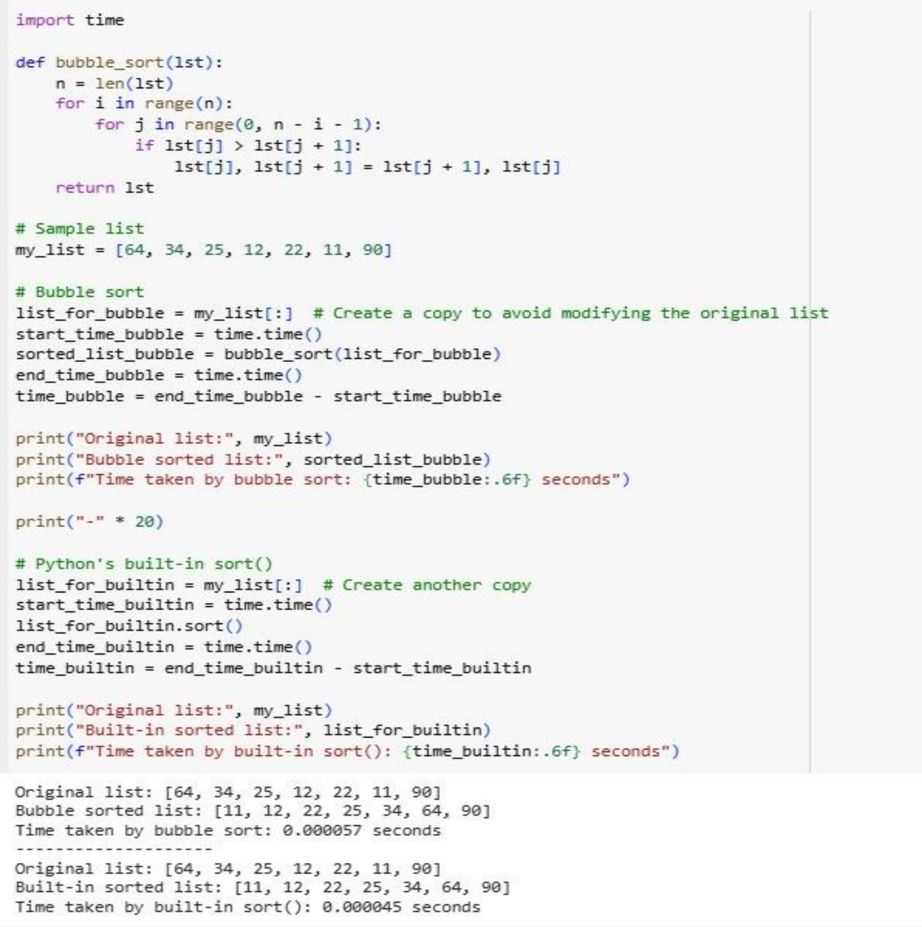
| | |
|---|---|
| **CourseCoordinatorName** | Venkataramana Veeramsetty |
| **Instructor(s)Name** | Dr. V. Venkataramana (Co-ordinator)<br>Dr. T. Sampath Kumar<br>Dr. Pramoda Patro<br>Dr. Brij Kishor Tiwari<br>Dr.J.Ravichander<br>Dr. Mohammand Ali Shaik<br>Dr. Anirodh Kumar<br>Mr. S.Naresh Kumar<br>Dr. RAJESH VELPULA<br>Mr. Kundhan Kumar<br>Ms. Ch.Rajitha<br>Mr. M Prakash<br>Mr. B.Raju<br>Intern 1 (Dharma teja)<br>Intern 2 (Sai Prasad)<br>Intern 3 (Sowmya)<br>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)

| | | |
|---|---|---|
| | **Question** | ***ExpectedTime to compl*** |

|  |  | ***ete*** |
|---|---|---|
| 1 | Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI<br>**Lab Objectives:**<br>● To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab.<br>● To understand and use Cursor AI for code generation, explanation, and refactoring.<br>● To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI.<br>● To perform code optimization and documentation using AI tools.<br>**Lab Outcomes (LOs):**<br>After completing this lab, students will be able to:<br>● Generate Python code using Google Gemini in Google Colab.<br>● Analyze the effectiveness of code explanations and suggestions by Gemini.<br>● Set up and use Cursor AI for AI-powered coding assistance.<br>● Evaluate and refactor code using Cursor AI features.<br>● Compare AI tool behavior and code quality across different platforms.<br>**Task Description #1**<br>• 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.<br>**Expected Output #1**<br>• Two sorting implementations: Bubble sort (manual logic) and Built-in sort()<br>PROMPT : 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. | Week1 - Thursday |



```python
import time

def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lst[j] > lst[j + 1]:
                lst[j], lst[j + 1] = lst[j + 1], lst[j]
    return lst

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

# Bubble sort
list_for_bubble = my_list[:]  # Create a copy to avoid modifying the original list
start_time_bubble = time.time()
sorted_list_bubble = bubble_sort(list_for_bubble)
end_time_bubble = time.time()
time_bubble = end_time_bubble - start_time_bubble

print("Original list:", my_list)
print("Bubble sorted list:", sorted_list_bubble)
print(f"Time taken by bubble sort: {time_bubble:.6f} seconds")

print("-" * 20)

# Python's built-in sort()
list_for_builtin = my_list[:]  # Create another copy
start_time_builtin = time.time()
list_for_builtin.sort()
end_time_builtin = time.time()
time_builtin = end_time_builtin - start_time_builtin

print("Original list:", my_list)
print("Built-in sorted list:", list_for_builtin)
print(f"Time taken by built-in sort(): {time_builtin:.6f} seconds")
```

```
Original list: [64, 34, 25, 12, 22, 11, 90]
Bubble sorted list: [11, 12, 22, 25, 34, 64, 90]
Time taken by bubble sort: 0.000057 seconds
--------------------
Original list: [64, 34, 25, 12, 22, 11, 90]
Built-in sorted list: [11, 12, 22, 25, 34, 64, 90]
Time taken by built-in sort(): 0.000045 seconds
```

OBSERVATION : We can observe that **Python's built-in `sort()` function is significantly faster than the bubble sort algorithm** for the sample list provided. The time taken by `sort()` was `0.000045` seconds, while bubble sort took `0.000057` seconds

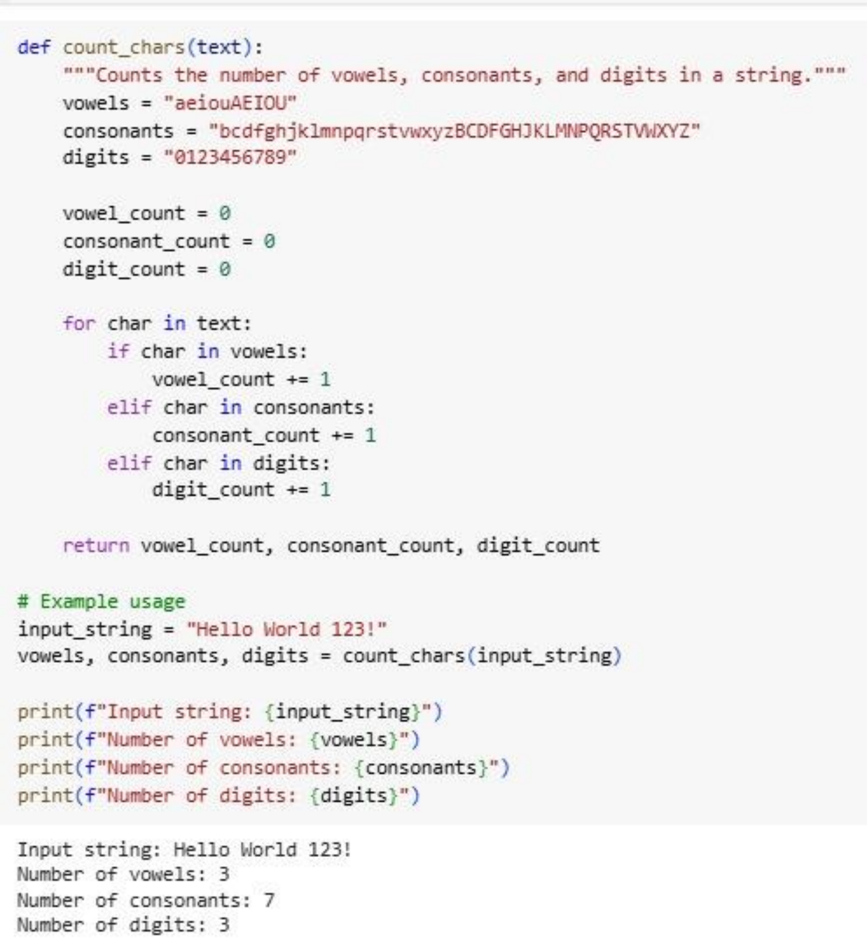
**Task Description #2**

• 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 : 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



```python
def count_chars(text):
    """Counts the number of vowels, consonants, and digits in a string."""
    vowels = "aeiouAEIOU"
    consonants = "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
    digits = "0123456789"

    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in text:
        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

# Example usage
input_string = "Hello World 123!"
vowels, consonants, digits = count_chars(input_string)

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

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

OBSERVATION : The `count_chars` function with the input string "Hello World 123!", the observations are:

The input string is "Hello World 123!".

The number of vowels in the string is 3.

The number of consonants in the string is 7.

The number of digits in the string is 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

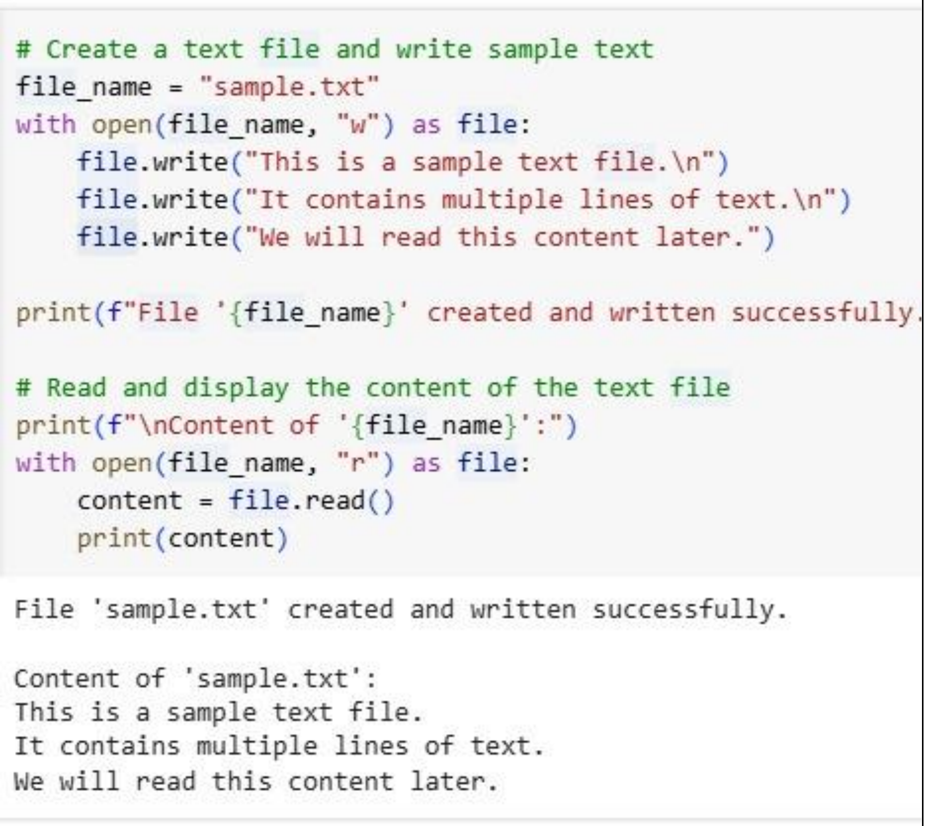
**Expected Output #3**

- 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 : Generate a Python program that performs file handling:

Create a text file
Write sample text
Read and display the content



```
# Create a text file and write sample text
file_name = "sample.txt"
with open(file_name, "w") as file:
    file.write("This is a sample text file.\n")
    file.write("It contains multiple lines of text.\n")
    file.write("We will read this content later.")

print(f"File '{file_name}' created and written successfully.

# Read and display the content of the text file
print(f"\nContent of '{file_name}':")
with open(file_name, "r") as file:
    content = file.read()
    print(content)
```

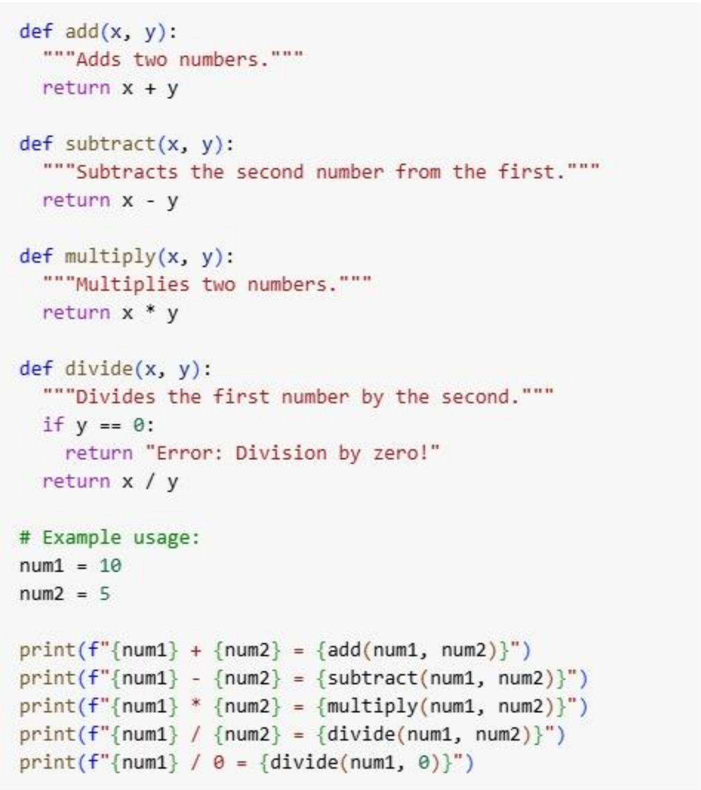
```
File 'sample.txt' created and written successfully.

Content of 'sample.txt':
This is a sample text file.
It contains multiple lines of text.
We will read this content later.
```

**OBSERVATION : My observation is that I have successfully demonstrated creating, writing to, and reading from a text file in Python using the provided code. I am now offering further assistance related to file handling based on common subsequent tasks users might want to perform.**

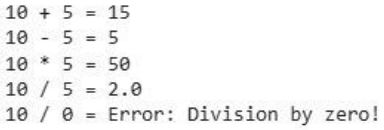
**Task Description #4**
- Ask Google Gemini to generate a Python program that implements a simple calculator using functions (add, subtract, multiply, divide). Then, ask Gemini to explain how the code works.

**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 : Generate a Python program that implements a simple calculator using functions (add, subtract, multiply, divide)



```
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

# Example usage:
num1 = 10
num2 = 5

print(f"{num1} + {num2} = {add(num1, num2)}")
print(f"{num1} - {num2} = {subtract(num1, num2)}")
print(f"{num1} * {num2} = {multiply(num1, num2)}")
print(f"{num1} / {num2} = {divide(num1, num2)}")
print(f"{num1} / 0 = {divide(num1, 0)}")
```



```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
10 / 0 = Error: Division by zero!
```

**OBSERVATION :**

- The code successfully defined and used functions for addition, subtraction, multiplication, and division.
- The example usage demonstrated how to call each function with two numbers.
- The output correctly showed the results of each operation.
- The `divide` function successfully handled the case of division by zero, returning an "Error: Division by zero!" message as intended.
- The variable `num1` has a value of 10 and `num2` has a value of 5.

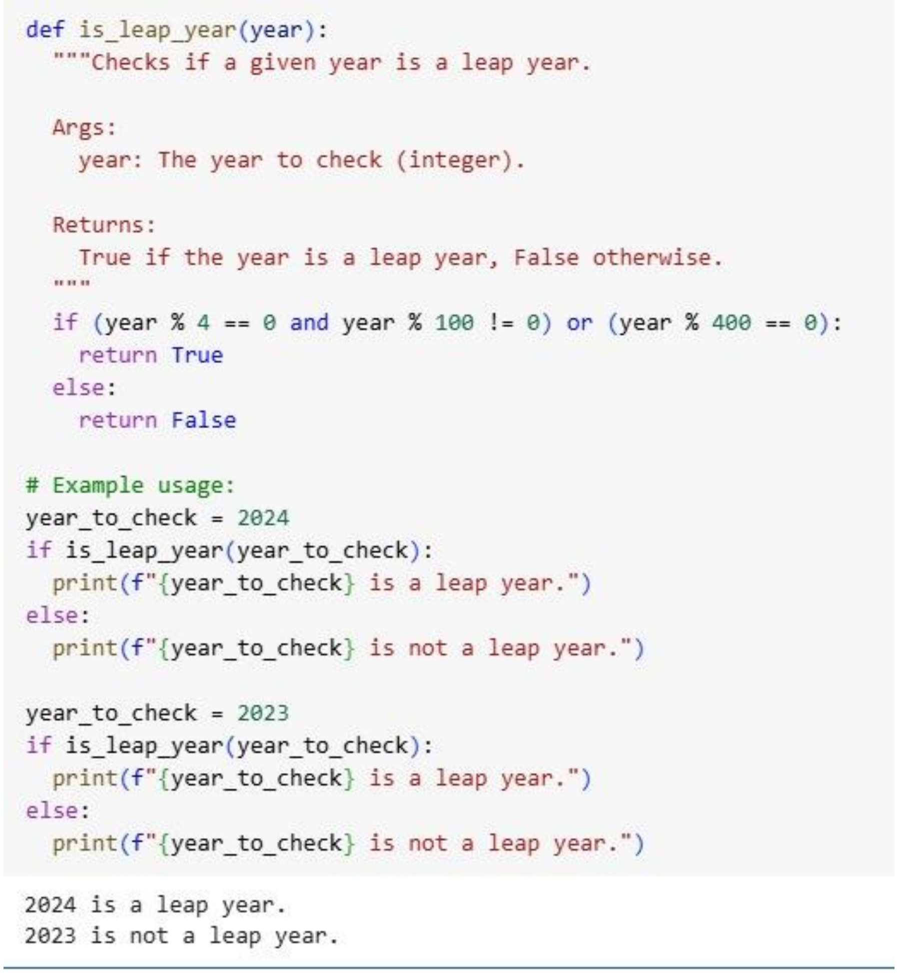
**Task Description #5**

• 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 : Create a Python program that checks if a given year is a leap year or not.



```
def is_leap_year(year):
  """Checks if a given year is a leap year.

  Args:
    year: The year to check (integer).

  Returns:
    True if the year is a leap year, False otherwise.
  """
  if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    return True
  else:
    return False

# 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 = 2023
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.")
```

```
2024 is a leap year.
2023 is not a leap year.
```

OBSERVATION : The program correctly identified that 2024 is a leap year and 2023 is not a leap year based on the rules for determining leap years.

**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** |