

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		Dr. Rishabh Mittal	
		Mr. S Naresh Kumar	
		Ms. B. Swathi	
		Dr. Sasanko Shekhar Gantayat	
		Mr. Md Sallauddin	
		Dr. Mathivanan	
		Mr. Y Srikanth	
		Ms. N Shilpa	
		Dr. Rishabh Mittal (Coordinator)	
		Dr. R. Prashant Kumar	
		Mr. Ankushavali MD	
		Mr. B Viswanath	
		Ms. Sujitha Reddy	
		Ms. A. Anitha	
		Ms. M.Madhuri	
		Ms. Katherashala Swetha	
		Ms. Velpula sumalatha	
		Mr. Bingi Raju	
Course Code	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/I	Regulation	R23
Date and Day of Assignment	Week 2 - Wednesday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number: 3.3(Present assignment number)/24(Total number of assignments)			
Q.No.	Question	Expected Time to complete	
1	<p><b>Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques</b></p> <p><b>Lab Objectives</b></p> <ul style="list-style-type: none"> <li>To explore and apply different levels of prompt examples in AI-assisted code generation</li> <li>To understand how zero-shot, one-shot, and few-shot prompting affect AI output quality</li> <li>To evaluate the impact of context richness and example quantity on AI performance</li> <li>To build awareness of prompt strategy effectiveness for different problem types</li> </ul>	Week2 - Wednesday	

## Lab Outcomes (LOs)

After completing this lab, students will be able to:

- Use zero-shot prompting to instruct AI with minimal context
- Use one-shot prompting with a single example to guide AI code generation
- Apply few-shot prompting using multiple examples to improve AI responses
- Compare AI outputs across different prompting strategies

## Task 1: Zero-Shot Prompting – Leap Year Check

### Scenario

Zero-shot prompting involves giving instructions without providing examples.

### Task Description

Use zero-shot prompting to instruct an AI tool to generate a Python function that:

- Accepts a year as input
- Checks whether the given year is a leap year
- Returns an appropriate result

**Note:** No input-output examples should be provided in the prompt.

### Expected Output

- AI-generated leap year checking function
- Correct logical conditions
- Sample input and output
- Screenshot of AI-generated response (if required)

The screenshot displays a Jupyter Notebook environment with two open files. The top file, 'Untitled1.ipynb', contains a prompt for an AI model to generate a Python function. The prompt is: 'representing a year as input. The function should determine whether the given year is a leap year using correct logical conditions and return an appropriate result. Also include a brief explanation of the logic used, along with a sample input and its corresponding output. Do not use any input-output examples inside the function definition'. The bottom file, '1602-ai-4.3.ipynb', shows the AI-generated Python code for the leap year checker function. The code defines a function 'is\_leap\_year(year)' that returns a boolean value based on the following logic: 'True if the year is a leap year, False otherwise. ... return year % 4 == 0 and year % 100 != 0 or (year % 400 == 0)'. The code also includes sample inputs and outputs: 'sample\_year = 2024', 'sample\_year\_2 = 1999', 'sample\_year\_3 = 1900', and 'sample\_year\_4 = 2023'. The AI-generated code is as follows: 

```
def is_leap_year(year):  
    """  
    Determine whether the given year is a leap year.  
    Args:  
        year: An integer representing the year.  
    Returns:  
        True if the year is a leap year, False otherwise.  
    """  
    return year % 4 == 0 and year % 100 != 0 or (year % 400 == 0)  
  
# Sample input and output  
sample_year = 2024  
print(f'is_leap_year({sample_year}) = {is_leap_year(sample_year)}')  
  
sample_year_2 = 1999  
print(f'is_leap_year({sample_year_2}) = {is_leap_year(sample_year_2)}')  
  
sample_year_3 = 1900  
print(f'is_leap_year({sample_year_3}) = {is_leap_year(sample_year_3)}')  
  
sample_year_4 = 2023  
print(f'is_leap_year({sample_year_4}) = {is_leap_year(sample_year_4)}')
```

## CODE EXPLANATION:

- A year is a leap year if:
  - It is divisible by 4 **and not** divisible by 100
  - OR divisible by 400
- Zero-shot prompting means **no examples were given in the prompt**

### Task 2: One-Shot Prompting – Centimeters to Inches Conversion

#### Scenario

One-shot prompting guides AI using a single example.

#### Task Description

Use one-shot prompting by providing one input-output example to generate a Python function that:

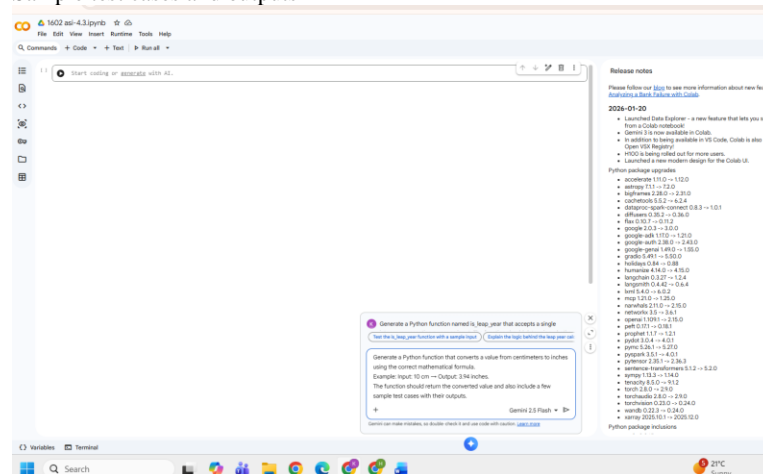
- Converts centimeters to inches
- Uses the correct mathematical formula

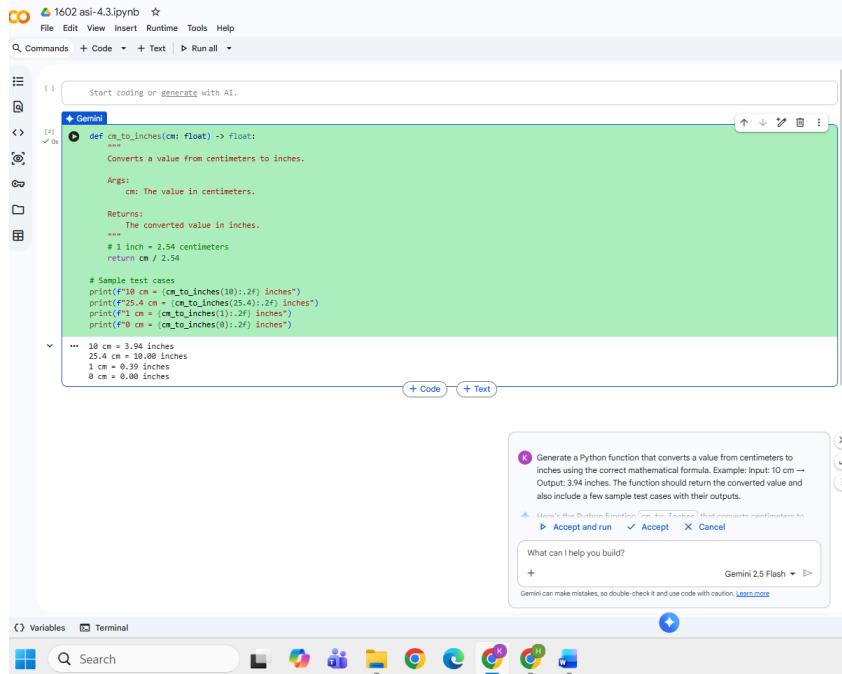
#### Example provided in prompt:

Input: 10 cm → Output: 3.94 inches

#### Expected Output

- Python function with correct conversion logic
- Accurate calculation
- Sample test cases and outputs





```
def cm_to_inches(cm: float) -> float:
    """
    Converts a value from centimeters to inches.

    Args:
        cm: The value in centimeters.

    Returns:
        The converted value in inches.

    """
    # 1 inch = 2.54 centimeters
    return cm / 2.54

# Sample test cases
print(f"10 cm = {cm_to_inches(10):.2f} inches")
print(f"25.4 cm = {cm_to_inches(25.4):.2f} inches")
print(f"1 cm = {cm_to_inches(1):.2f} inches")
print(f"0 cm = {cm_to_inches(0):.2f} inches")
```

10 cm = 3.94 inches  
25.4 cm = 10.00 inches  
1 cm = 0.39 inches  
0 cm = 0.00 inches

## CODE EXPLANATION:

- Formula used:  
**1 inch = 2.54 cm**
- One-shot prompting uses **one example** (10 cm → 3.94 inches)

### Task 3: Few-Shot Prompting – Name Formatting

#### Scenario

Few-shot prompting improves accuracy by providing multiple examples.

#### Task Description

Use few-shot prompting with 2–3 examples to generate a Python function that:

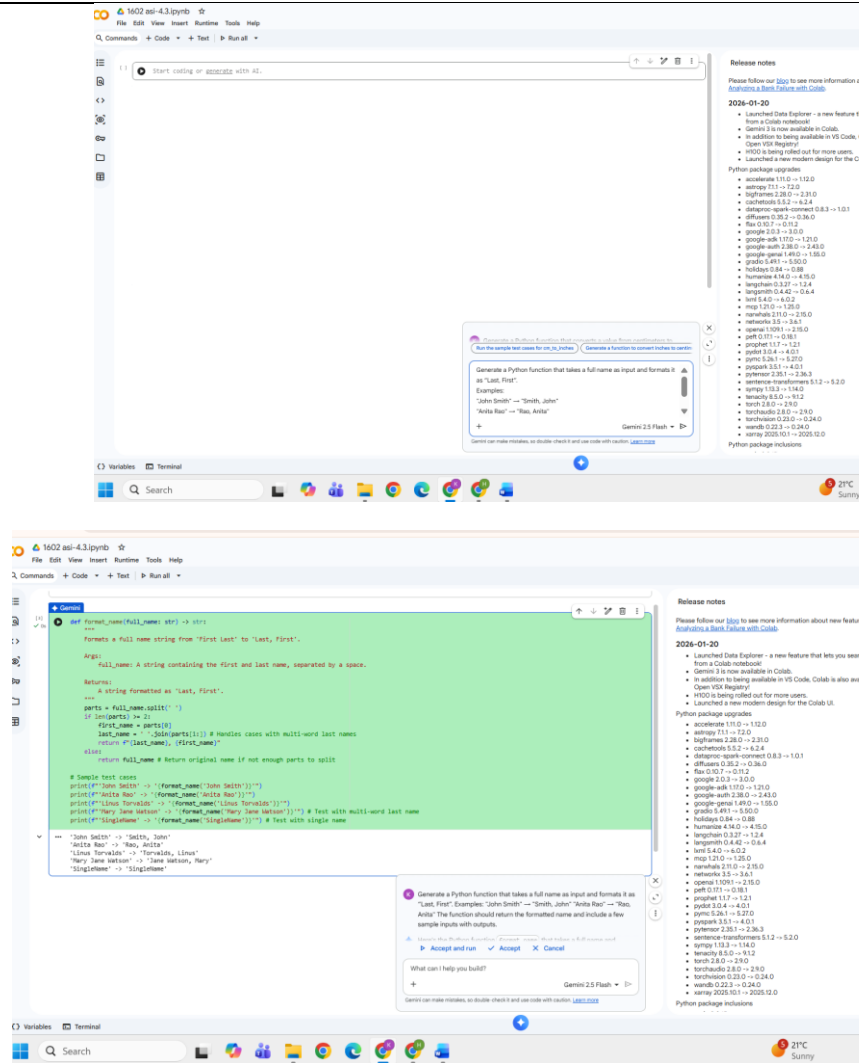
- Accepts a full name as input
- Formats it as “Last, First”

#### Example formats:

- "John Smith" → "Smith, John"
- "Anita Rao" → "Rao, Anita"

#### Expected Output

- Well-structured Python function
- Output strictly following example patterns
- Correct handling of names
- Sample inputs and outputs



## CODE EXPLANATION:

- The name is split into first and last parts
- Output follows the pattern:  
**Last, First**
- Few-shot prompting provides multiple examples

## Task 4: Comparative Analysis – Zero-Shot vs Few-Shot

### Scenario

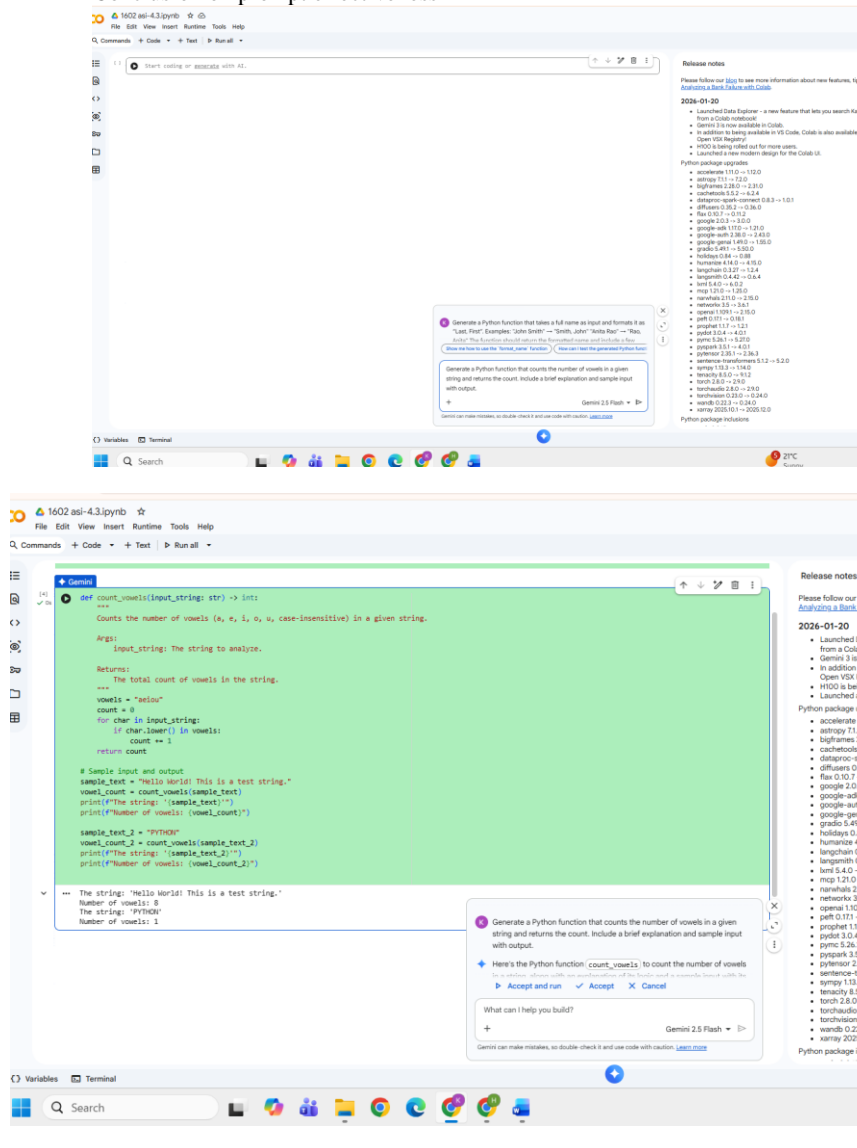
Different prompt strategies may produce different code quality.

### Task Description

- Use zero-shot prompting to generate a function that counts vowels in a string
- Use few-shot prompting for the same problem
- Compare both outputs based on:
  - Accuracy
  - Readability
  - Logical clarity

### Expected Output

- Two vowel-counting functions
- Comparison table or short reflection paragraph
- Conclusion on prompt effectiveness



The screenshot displays the Google Colab interface with a Python function for counting vowels in a string. The function is named `count_vowels` and takes an input string. It counts the number of vowels (a, e, i, o, u) in a case-insensitive manner. The function is tested with two sample inputs: "Hello World! This is a test string." and "PYTHON". The output shows the count of vowels for each string.

```
def count_vowels(input_string: str) -> int:
    """
    Counts the number of vowels (a, e, i, o, u, case-insensitive) in a given string.

    Args:
        input_string: The string to analyze.

    Returns:
        The total count of vowels in the string.
    """
    vowels = "aeiou"
    count = 0
    for char in input_string:
        if char.lower() in vowels:
            count += 1
    return count

# Sample input and output
sample_text = "Hello World! This is a test string."
vowel_count = count_vowels(sample_text)
print(f"The string: '{sample_text}'")
print(f"Number of vowels: {vowel_count}")

sample_text_2 = "PYTHON"
vowel_count_2 = count_vowels(sample_text_2)
print(f"The string: '{sample_text_2}'")
print(f"Number of vowels: {vowel_count_2}")
```

Output:

```
--- The string: 'Hello World! This is a test string.'
Number of vowels: 8
The string: 'PYTHON'
Number of vowels: 1
```

# CODE EXPLANATION:

## Task 5: Few-Shot Prompting – File Handling

### Scenario

File processing requires clear logical understanding.

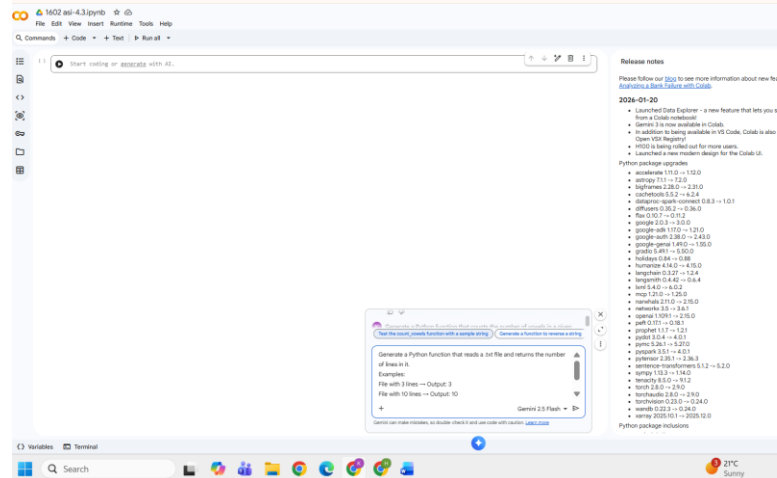
### Task Description

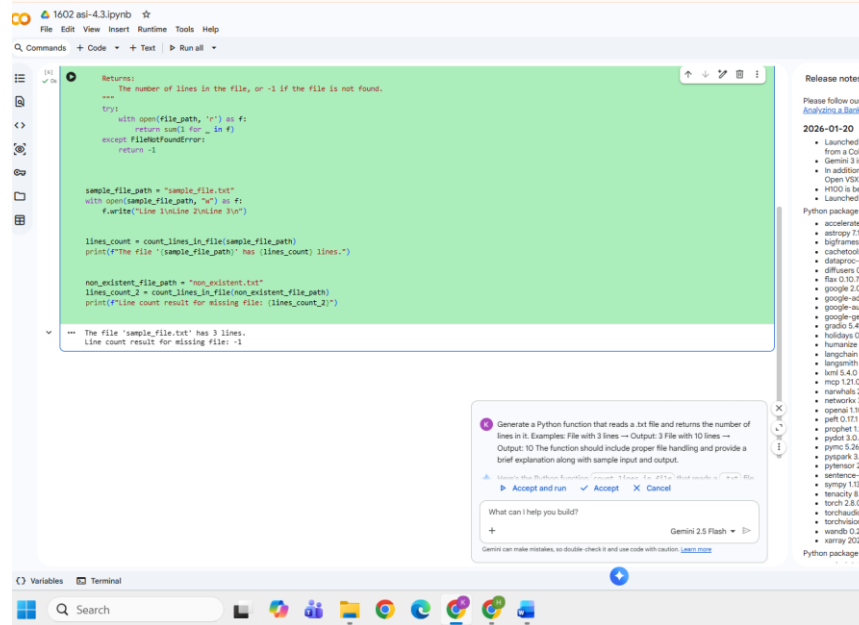
Use few-shot prompting to generate a Python function that:

- Reads a .txt file
- Counts the number of lines in the file
- Returns the line count

### Expected Output

- Working Python file-processing function
- Correct line count
- Sample .txt input and output
- AI-assisted logic explanation





```
def count_lines_in_file(file_path):  
    """Returns:  
    The number of lines in the file, or -1 if the file is not found.  
    """  
    try:  
        with open(file_path, "r") as f:  
            return sum(1 for _ in f)  
    except FileNotFoundError:  
        return -1  
  
sample_file_path = "sample_file.txt"  
with open(sample_file_path, "r") as f:  
    f.write("Line 1\nLine 2\nLine 3\n")  
  
lines_count = count_lines_in_file(sample_file_path)  
print(f"The file '{sample_file_path}' has {lines_count} lines.")  
  
non_existent_file_path = "non_existent.txt"  
lines_count_2 = count_lines_in_file(non_existent_file_path)  
print(f"Line count result for missing file: {lines_count_2}")  
  
---  
The file 'sample_file.txt' has 3 lines.  
Line count result for missing file: -1
```

Generate a Python function that reads a .txt file and returns the number of lines in it. Examples: File with 3 lines → Output: 3 File with 10 lines → Output: 10 The function should include proper file handling and provide a brief explanation along with sample input and output.

What can I help you build?

## CODE EXPLANATION:

- Opens the file in read mode
- Counts lines using `sum(1 for _ in f)`
- Returns -1 if file does not exist
- Few-shot prompting uses **multiple example outputs**

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