| | |
|---|---|
| **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** | Week2 - Wednesday | **Time(s)** | |
| **Duration** | 2 Hours | **Applicableto Batches** | |

**AssignmentNumber:4.3**(Present assignment number)/**24**(Total number of assignments)

| **Q.No.** | **Question** | ***ExpectedTime to complete*** |
|---|---|---|
| 1 | Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques<br>**Lab Objectives:**<br>● To explore and apply different levels of prompt examples in AI-assisted code | Week2 - Wednesday |

generation.

- To understand how zero-shot, one-shot, and few-shot prompting affect AI output quality.
- To evaluate the impact of context richness and example quantity on AI performance.
- To build awareness of prompt strategy effectiveness for different problem types.

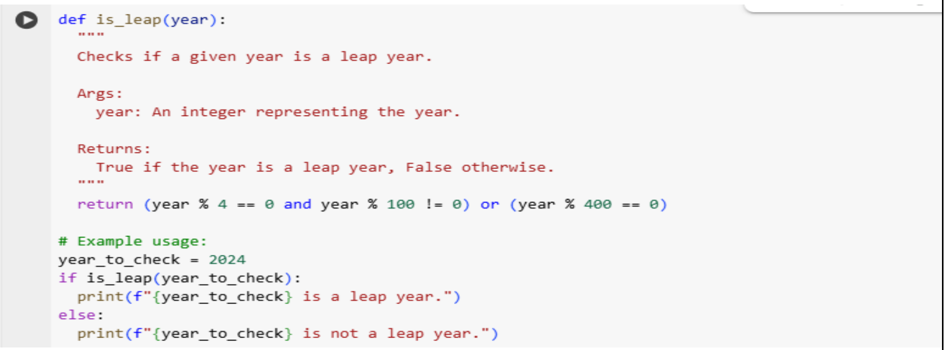
**Lab Outcomes (LOs):**

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 the three prompting strategies.

**Task Description#1**

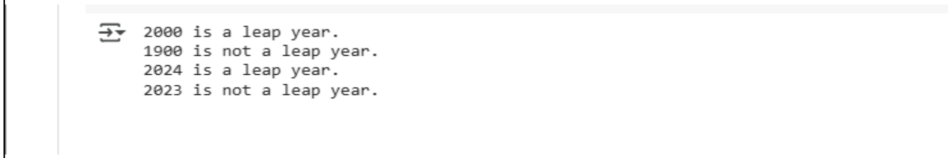
- Zero-shot: Prompt AI to write a function that checks whether a given year is a leap year.
- **Prompt : Write a function that checks whether a given year is a leap year**



```python
def is_leap(year):
  """
  Checks if a 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)

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

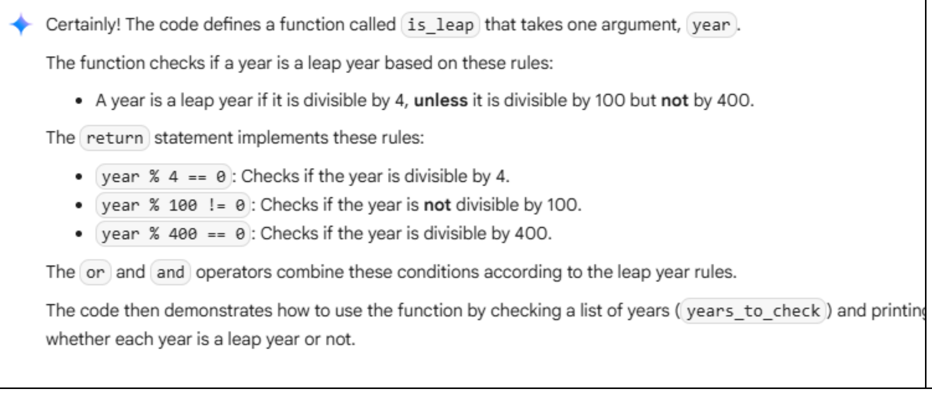
**Expected Output#1**

- AI-generated function with no examples provided



```
2000 is a leap year.
1900 is not a leap year.
2024 is a leap year.
2023 is not a leap year.
```

- **Explanation :**



Certainly! The code defines a function called `is_leap` that takes one argument, `year`.

The function checks if a year is a leap year based on these rules:

- A year is a leap year if it is divisible by 4, **unless** it is divisible by 100 but **not** by 400.

The `return` statement implements these rules:

- `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.

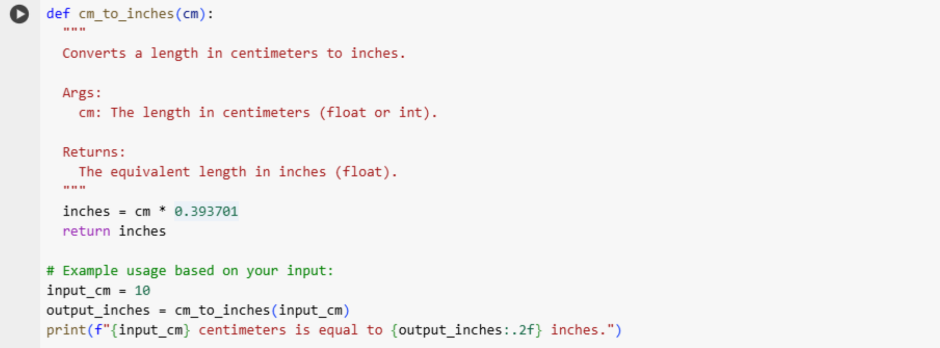
The `or` and `and` operators combine these conditions according to the leap year rules.

The code then demonstrates how to use the function by checking a list of years (`years_to_check`) and printin whether each year is a leap year or not.

**Task Description#2**

- One-shot: Give one input-output example to guide AI in writing a function that converts centimeters to inches

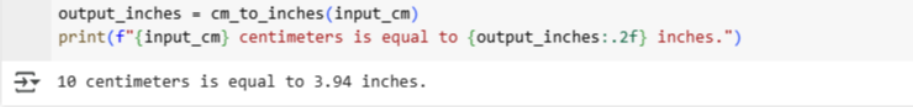
- **Prompt : Give one input – output example to guide AI in writing a function that converts centimetrs into inches.**



```python
def cm_to_inches(cm):
  """
  Converts a length in centimeters to inches.

  Args:
    cm: The length in centimeters (float or int).

  Returns:
    The equivalent length in inches (float).
  """
  inches = cm * 0.393701
  return inches

# Example usage based on your input:
input_cm = 10
output_inches = cm_to_inches(input_cm)
print(f"{input_cm} centimeters is equal to {output_inches:.2f} inches.")
```

**Expected Output#2**

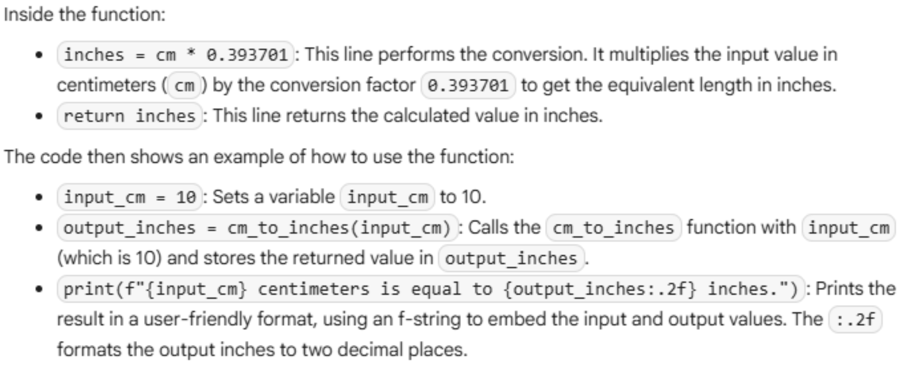
- Function with correct conversion logic



```python
output_inches = cm_to_inches(input_cm)
print(f"{input_cm} centimeters is equal to {output_inches:.2f} inches.")
```

```
10 centimeters is equal to 3.94 inches.
```

- **Explanation :**



Inside the function:

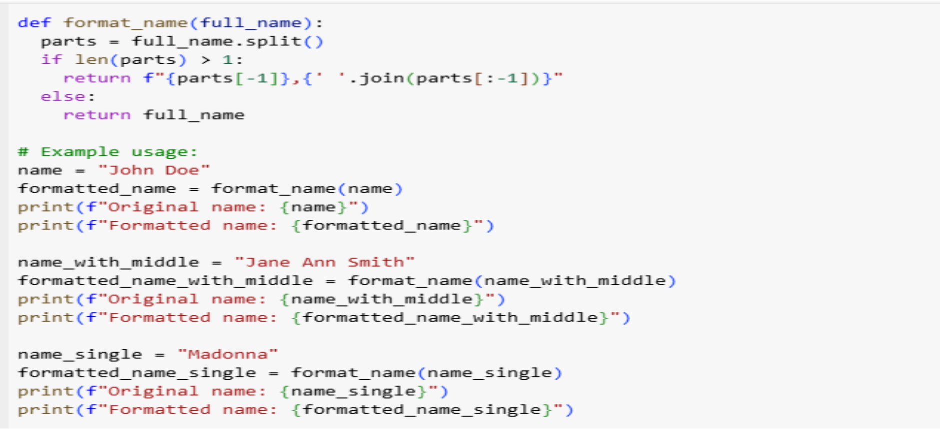
- `inches = cm * 0.393701`: This line performs the conversion. It multiplies the input value in centimeters (`cm`) by the conversion factor `0.393701` to get the equivalent length in inches.
- `return inches`: This line returns the calculated value in inches.

The code then shows an example of how to use the function:

- `input_cm = 10`: Sets a variable `input_cm` to 10.
- `output_inches = cm_to_inches(input_cm)`: Calls the `cm_to_inches` function with `input_cm` (which is 10) and stores the returned value in `output_inches`.
- `print(f"{input_cm} centimeters is equal to {output_inches:.2f} inches.")`: Prints the result in a user-friendly format, using an f-string to embed the input and output values. The `:.2f` formats the output inches to two decimal places.

**Task Description#3**

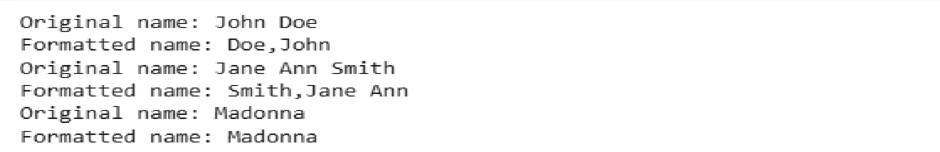
- Few-shot: Provide 2–3 examples to generate a function that formats full names as "Last, First".
- **Prompt : Generate a function that formats full names as "Last,First".**



```python
def format_name(full_name):
  parts = full_name.split()
  if len(parts) > 1:
    return f"{parts[-1]},{' '.join(parts[:-1])}"
  else:
    return full_name

# Example usage:
name = "John Doe"
formatted_name = format_name(name)
print(f"Original name: {name}")
print(f"Formatted name: {formatted_name}")

name_with_middle = "Jane Ann Smith"
formatted_name_with_middle = format_name(name_with_middle)
print(f"Original name: {name_with_middle}")
print(f"Formatted name: {formatted_name_with_middle}")

name_single = "Madonna"
formatted_name_single = format_name(name_single)
print(f"Original name: {name_single}")
print(f"Formatted name: {formatted_name_single}")
```

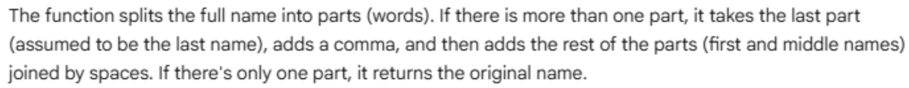
**Expected Output#3**

- Well-structured function respecting the examples



```
Original name: John Doe
Formatted name: Doe,John
Original name: Jane Ann Smith
Formatted name: Smith,Jane Ann
Original name: Madonna
Formatted name: Madonna
```

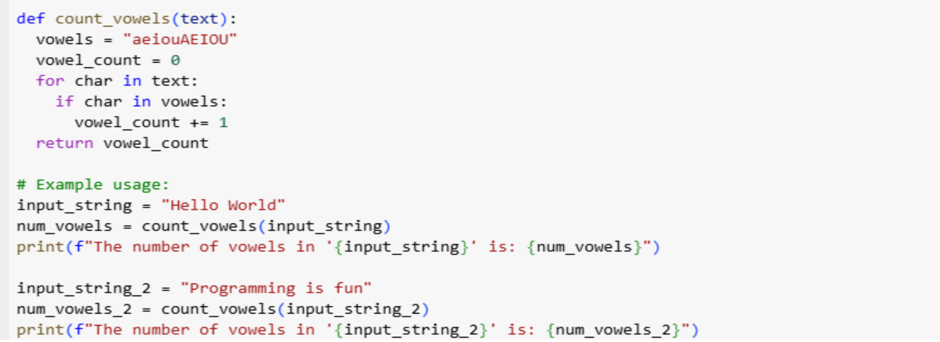
- **Explanation :**



The function splits the full name into parts (words). If there is more than one part, it takes the last part (assumed to be the last name), adds a comma, and then adds the rest of the parts (first and middle names) joined by spaces. If there's only one part, it returns the original name.

**Task Description#4**

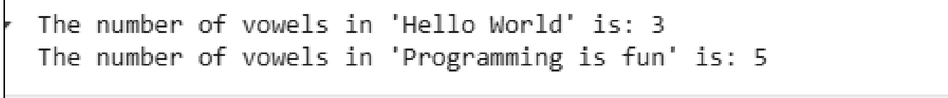
- Compare zero-shot and few-shot prompts for writing a function that counts the number of vowels in a string.
- **Prompt : Write a function that counts the number of vowels in a given string.**



```python
def count_vowels(text):
  vowels = "aeiouAEIOU"
  vowel_count = 0
  for char in text:
    if char in vowels:
      vowel_count += 1
  return vowel_count

# Example usage:
input_string = "Hello World"
num_vowels = count_vowels(input_string)
print(f"The number of vowels in '{input_string}' is: {num_vowels}")

input_string_2 = "Programming is fun"
num_vowels_2 = count_vowels(input_string_2)
print(f"The number of vowels in '{input_string_2}' is: {num_vowels_2}")
```

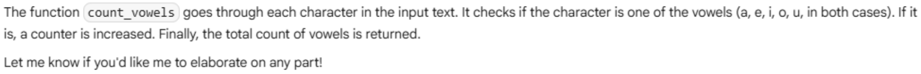
**Expected Output#4**

- Functional output and comparative reflection



```
The number of vowels in 'Hello World' is: 3
The number of vowels in 'Programming is fun' is: 5
```

- **Explanation :**



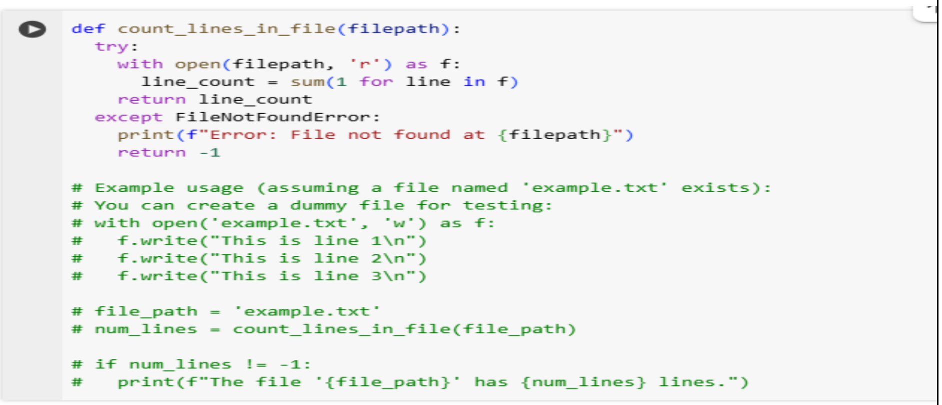
The function `count_vowels` goes through each character in the input text. It checks if the character is one of the vowels (a, e, i, o, u, in both cases). If it is, a counter is increased. Finally, the total count of vowels is returned.

Let me know if you'd like me to elaborate on any part!

**Task Description#5**

- Use few-shot prompting to generate a function that reads a .txt file and returns the number of lines.

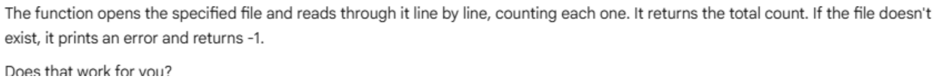
- **Prompt : Generate a function that reads a txt.file and returns the number of lines**



```
def count_lines_in_file(filepath):
  try:
    with open(filepath, 'r') as f:
      line_count = sum(1 for line in f)
    return line_count
  except FileNotFoundError:
    print(f"Error: File not found at {filepath}")
    return -1

# Example usage (assuming a file named 'example.txt' exists):
# You can create a dummy file for testing:
# with open('example.txt', 'w') as f:
#   f.write("This is line 1\n")
#   f.write("This is line 2\n")
#   f.write("This is line 3\n")

# file_path = 'example.txt'
# num_lines = count_lines_in_file(file_path)

# if num_lines != -1:
#   print(f"The file '{file_path}' has {num_lines} lines.")
```

**Expected Output#5**

- Working file-processing function with AI-guided logic



The function opens the specified file and reads through it line by line, counting each one. It returns the total count. If the file doesn't exist, it prints an error and returns -1.

Does that work for you?

**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 |
|---|---|
| Zero Shot (Task #1) | 0.5 |
| One Shot (Task#2) | 0.5 |
| Few Shot (Task#3 & Task #5) | 1.0 |
| Comparison (Task#4) | 0.5 |
| **Total** | **2.5 Marks** |