

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
2303A53012(B-46)			
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		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	Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques Lab Objectives <ul style="list-style-type: none"> To explore and apply different levels of prompt examples in AI-assisted code generation To understand how zero-shot, one-shot, and few-shot prompting affect AI output 		Week2 - Wednesday

	<p>quality</p> <ul style="list-style-type: none">• To evaluate the impact of context richness and example quantity on AI performance• To build awareness of prompt strategy effectiveness for different problem types <p>Lab Outcomes (LOs) After completing this lab, students will be able to:</p> <ul style="list-style-type: none">• 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	
	<p>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:</p> <ul style="list-style-type: none">• Accepts a year as input• Checks whether the given year is a leap year• Returns an appropriate result <p>Note: No input-output examples should be provided in the prompt. Expected Output</p> <ul style="list-style-type: none">• AI-generated leap year checking function• Correct logical conditions• Sample input and output• Screenshot of AI-generated response (if required) <pre>Generate a Python function that checks whether a given year is a leap year. The function should accept a year as input and return True if it's a leap year, False otherwise. Include the correct leap year logic in the function.</pre> <pre>1 def is_leap_year(year): 2 # A year is a leap year if: 3 # 1. It is divisible by 4 AND 4 # 2. If it is divisible by 100, it must also be divisible by 400 5 6 if year % 400 == 0: 7 return True 8 elif year % 100 == 0: 9 return False 10 elif year % 4 == 0: 11 return True 12 else: 13 return False</pre>	

	<table><tr><th>Year</th><th>Expected Result</th><th>Reason</th></tr><tr><td>2000</td><td>True (Leap)</td><td>Divisible by 400</td></tr><tr><td>1900</td><td>False (Not Leap)</td><td>Divisible by 100 but not 400</td></tr><tr><td>2024</td><td>True (Leap)</td><td>Divisible by 4, not divisible by 100</td></tr><tr><td>2023</td><td>False (Not Leap)</td><td>Not divisible by 4</td></tr><tr><td>2004</td><td>True (Leap)</td><td>Divisible by 4, not divisible by 100</td></tr><tr><td>2100</td><td>False (Not Leap)</td><td>Divisible by 100 but not 400</td></tr></table> <pre>24 25 # Test cases 26 print(is_leap_year(2000)) # Output: True 27 print(is_leap_year(1900)) # Output: False 28 print(is_leap_year(2024)) # Output: True 29 print(is_leap_year(2023)) # Output: False 30 print(is_leap_year(2004)) # Output: True 31 print(is_leap_year(2100)) # Output: False</pre> <p>OUTPUT PROBLEMS PORTS SPELL CHECKER DEBUG CONSOLE</p> <pre>PS C:\Users\shaik\Downloads\Projects-source_code\pyth True False True False True False PS C:\Users\shaik\Downloads\Projects-source_code\pyth</pre>	Year	Expected Result	Reason	2000	True (Leap)	Divisible by 400	1900	False (Not Leap)	Divisible by 100 but not 400	2024	True (Leap)	Divisible by 4, not divisible by 100	2023	False (Not Leap)	Not divisible by 4	2004	True (Leap)	Divisible by 4, not divisible by 100	2100	False (Not Leap)	Divisible by 100 but not 400	
Year	Expected Result	Reason																					
2000	True (Leap)	Divisible by 400																					
1900	False (Not Leap)	Divisible by 100 but not 400																					
2024	True (Leap)	Divisible by 4, not divisible by 100																					
2023	False (Not Leap)	Not divisible by 4																					
2004	True (Leap)	Divisible by 4, not divisible by 100																					
2100	False (Not Leap)	Divisible by 100 but not 400																					
	<p>Task 2: One-Shot Prompting – Centimeters to Inches Conversion</p> <p>Scenario One-shot prompting guides AI using a single example.</p> <p>Task Description Use one-shot prompting by providing one input-output example to generate a Python function that:</p> <ul style="list-style-type: none">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

Generate a Python function that converts centimeters to inches.

Example:

Input: 10 cm → Output: 3.94 inches

The function should accept a value in centimeters and return the equivalent

value in inches with appropriate rounding. Include proper documentation.

```
def cm_to_inches(cm):
    """
    Convert centimeters to inches.

    Args:
        cm (float): The value in centimeters to convert

    Returns:
        float: The equivalent value in inches, rounded to 2 decimal places

    Example:
        >>> cm_to_inches(10)
        3.94
    """
    # Conversion factor: 1 inch = 2.54 cm
    # Therefore: inches = cm / 2.54
    inches = cm / 2.54
    return round(inches, 2)
```

Inches = Centimeters ÷ 2.54

Running the Code

```
python

# Test cases
print(cm_to_inches(10))
print(cm_to_inches(2.54))
print(cm_to_inches(5))
print(cm_to_inches(30))
print(cm_to_inches(1))
print(cm_to_inches(100))
print(cm_to_inches(50))
```

Expected Output

```
3.94
1.0
1.97
11.81
0.39
39.37
19.69
```

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

	<ul style="list-style-type: none">• Output strictly following example patterns• Correct handling of names• Sample inputs and outputs <div><p>Generate a Python function that formats full names from "First Last" format to "Last, First" format.</p><p>Examples:</p><ol style="list-style-type: none">1. "John Smith" → "Smith, John"2. "Anita Rao" → "Rao, Anita"3. "Maria Garcia Lopez" → "Garcia Lopez, Maria"<p>The function should accept a full name as input and return the formatted name with the last name followed by a comma and the first name. Handle names with multiple parts appropriately.</p></div>	
--	---	--

```

1  def format_name(full_name):
2      """
3      Format a full name from "First Last" to "Last, First" format.
4
5      Args:
6          full_name (str): The full name in "First Last" format
7
8      Returns:
9          str: The formatted name in "Last, First" format
10
11     Examples:
12         >>> format_name("John Smith")
13         'Smith, John'
14         >>> format_name("Anita Rao")
15         'Rao, Anita'
16         >>> format_name("Maria Garcia Lopez")
17         'Garcia Lopez, Maria'
18     """
19     # Split the full name into parts
20     parts = full_name.strip().split()
21
22     if len(parts) < 2:
23         return full_name # Return as-is if only one name part
24
25     # First name is the first part
26     first_name = parts[0]
27
28     # Last name is everything after the first part
29     last_name = ' '.join(parts[1:])
30
31     # Return formatted as "Last, First"
32     return f"{last_name}, {first_name}"
33
34 # Test cases
35 print(format_name("John Smith"))
36 print(format_name("Anita Rao"))
37 print(format_name("Maria Garcia Lopez"))
38 print(format_name("William Henry Gates"))
39 print(format_name("Thomas Jefferson"))
40 print(format_name("Rosa Parks"))
41 print(format_name("Albert Einstein"))
42 print(format_name("Jane Austen"))
43 print(format_name("Vincent Van Gogh"))
44 print(format_name("Leonardo da Vinci"))
45

```

	<div> <div> OUTPUT PROBLEMS PORTS SPELL CHECKER DEBUG CONSOLE TERMINAL </div> <div> PS C:\Users\shaik\Downloads\Projects-source_code\python-PD> Smith, John Rao, Anita Garcia Lopez, Maria Henry Gates, William Jefferson, Thomas Parks, Rosa Einstein, Albert Austen, Jane Van Gogh, Vincent da Vinci, Leonardo PS C:\Users\shaik\Downloads\Projects-source_code\python-PD> </div> </div>	
	<p>Task 4: Comparative Analysis – Zero-Shot vs Few-Shot Scenario</p> <p>Different prompt strategies may produce different code quality.</p> <p>Task Description</p> <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> ○ Accuracy ○ Readability ○ Logical clarity <p>Expected Output</p> <ul style="list-style-type: none"> • Two vowel-counting functions • Comparison table or short reflection paragraph • Conclusion on prompt effectiveness <div> <div>Zero-Shot Prompt</div> <div> Prompt (Zero-Shot): "Write a function that counts the number of vowels in a given string </div> <div> <pre>def count_vowels(text): vowels = "aeiouAEIOU" count = 0 for char in text: if char in vowels: count += 1 return count</pre> </div> </div>	

	<h2>Few-Shot Prompt</h2> <p>Prompt (Few-Shot): "Here are some examples of similar problems: Example 1: Input: 'hello' Output: 2 Example 2: Input: 'PYTHON' Output: 1 Now, write a function that counts the number of vowels in a given string."</p> <pre>def count_vowels(text): return sum(1 for char in text.lower() if char in 'aeiou')</pre> <table> <caption>Comparison Table</caption> <thead> <tr> <th>Criteria</th><th>Zero-Shot Output</th><th>Few-Shot Output</th></tr> </thead> <tbody> <tr> <td>Accuracy</td><td>Correct for all cases</td><td>Correct for all cases</td></tr> <tr> <td>Readability</td><td>Very clear and beginner-friendly</td><td>Compact but slightly advanced</td></tr> <tr> <td>Logical Clarity</td><td>Step-by-step logic</td><td>Uses Pythonic logic (generator)</td></tr> <tr> <td>Efficiency</td><td>Moderate</td><td>Slightly more efficient</td></tr> </tbody> </table> <p>Zero-shot prompting produces correct and easy-to-understand code, making it suitable for beginner instructional purposes. Few-shot prompting results in more concise and optimized code by learning from provided examples. Overall, few-shot prompting is more effective when code quality, efficiency, and conciseness are important, while zero-shot prompting is better for clarity and simplicity.</p>	Criteria	Zero-Shot Output	Few-Shot Output	Accuracy	Correct for all cases	Correct for all cases	Readability	Very clear and beginner-friendly	Compact but slightly advanced	Logical Clarity	Step-by-step logic	Uses Pythonic logic (generator)	Efficiency	Moderate	Slightly more efficient	
Criteria	Zero-Shot Output	Few-Shot Output															
Accuracy	Correct for all cases	Correct for all cases															
Readability	Very clear and beginner-friendly	Compact but slightly advanced															
Logical Clarity	Step-by-step logic	Uses Pythonic logic (generator)															
Efficiency	Moderate	Slightly more efficient															
	<p>Task 5: Few-Shot Prompting – File Handling</p> <p>Scenario File processing requires clear logical understanding.</p> <p>Task Description Use few-shot prompting to generate a Python function that:</p> <ul style="list-style-type: none"> • Reads a .txt file • Counts the number of lines in the file • Returns the line count <p>Expected Output</p> <ul style="list-style-type: none"> • Working Python file-processing function • Correct line count • Sample .txt input and output • AI-assisted logic explanation 																

Few-Shot Prompt:

Example 1:

Task: Count the number of characters in a text file

Output: Total characters returned correctly

Example 2:

Task: Count the number of words in a text file

Output: Total words returned correctly

Now, write a Python function that reads a `.txt` file, counts the number of lines in the file, and returns the line count.

```
def count_lines(file_path):  
    try:  
        with open(file_path, 'r') as file:  
            lines = file.readlines()  
            return len(lines)  
    except FileNotFoundError:  
        return "File not found"
```

File name: `sample.txt`

csharp

Hello World

This is a sample file

Used for line counting

Python file handling

```
count = count_lines("sample.txt")  
print(count)
```

AI-Assisted Logic Explanation

The function uses Python's built-in `open()` method to read the text file.

`with` statement ensures the file is automatically closed after reading.

`readlines()` reads all lines from the file and stores them in a list.

`len()` counts how many lines are present in that list.

A `try-except` block is used to handle cases where the file does not exist, preventing runtime errors.