

# **SR UNIVERSITY**

## **AI ASSIST CODING**

### **LAB-4.3: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques**

**Name :**D.Prasanna Latha

**Pin No:**2503A51L37

**Batch:** 25BTCAICSB20

#### **Lab Objectives:**

- 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 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 #1:**

#### **Prompt:**

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

#### **Code Generated:**

The screenshot shows the Visual Studio Code interface. The left sidebar has icons for files, folders, and other workspace items. The main editor area displays the 'error.py' file with the following Python code:

```
C: > Users > Susmija > error.py > ...
1  # Prompt: Write a function that checks whether a given year is a leap year.
2  def is_leap_year(year):
3      """
4          Returns True if the given year is a leap year, False otherwise.
5      """
6      return year % 4 == 0 and (year % 100 != 0 or year% 400 == 0)
7  # ...existing code...
8  print(is_leap_year(2024)) # Example: Should print True
9  print(is_leap_year(1900)) # Example: Should print False
10 print(is_leap_year(2000)) # Example: Should print True
```

To the right of the editor, there is a 'CHAT' panel. It shows a message from the AI: "return year % 4 == 0 and (year % 100 != 0 or year% 400 == 0) y the output not showing". Below this, there are several status messages: "Used 1 reference", "Your code only defines the function; it does not call it or print anything.", "To see output, you need to call the function and print the result.", and "Add these lines at the end of your file: # ...existing code... print(is\_leap\_year(2024)) # print(is\_leap\_year(1900)) # print(is\_leap\_year(2000)) #".

#### **Output After executing Code:**

The screenshot shows the VS Code interface with the 'TERMINAL' tab selected. The terminal window displays the following command and its output:

```
PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe c:/Users/Susmija/error.py
● True
False
True
○ PS C:\Users\Susmija>
```

To the right of the terminal, there is a 'GPT-4.1' AI interface. It has a text input field with the command "print(is\_leap\_year(2000)) # E>". Below the input field, there is a message: "Now, when you run the script, you will see the output." A small AI icon is also visible in the bottom right corner of the AI interface.

## Your Observations:

### Code Functionality:

- **Purpose:** The code checks if a given year is a leap year.
- **Function:**
- `def is_leap_year(year: int) -> bool:`
- `return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)`
- This correctly implements the leap year rule:
  - A year divisible by 4 **and** not divisible by 100 **unless** divisible by 400 is a leap year.

### 1.The script uses:

`year = int(input("Enter a year: "))` to take user input and then prints:

- "`{year}` is a leap year." if the condition is true.
- Otherwise, "not a leap year"

### 2.Example Execution:

- Input: 2025
  - Output: 2025 is not a leap year.
1. Uses type hinting (`year: int -> bool`) – ✓ good practice.
  2. Uses f-strings for output – ✓ clean and readable.
  3. The function name `is_leap_year` is descriptive and appropriate.

## TASK #3:

### Prompt:

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

### Code Generated:

The screenshot shows a dark-themed instance of Visual Studio Code. The left sidebar contains icons for file operations like Open, Save, Find, and Run. The top menu bar includes File, Edit, Selection, View, Go, Run, and a Help section. The main workspace has several tabs open: "Student.py", "task-2.py", "def is\_leap\_year(year): Untitled-1", "error.py", and "task2.py". The "task2.py" tab is currently active, displaying the following Python code:

```
C: > Users > Susmija > Desktop > AI > task2.py > ...
1  # Function to convert centimeters to inches
2  def cm_to_inches(cm):
3      """
4          Converts centimeters to inches.
5      """
6      return cm / 2.54
7
8  # Example:
9  # Input: 10
10 # Output: 3.937
11 print(cm_to_inches(10)) #
```

To the right of the code editor, there is a Chat interface. A message from the AI says: "example to guide AI in writing a function that converts centimeters to inches. give the python code". Below the message, there is a code completion suggestion: "# Function to convert centimeter", followed by a snippet of code that matches the AI's response. The bottom of the screen shows the standard VS Code navigation bar with PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL tabs.

### Output After executing Code:

The screenshot shows a terminal window in a development environment. The terminal tab is selected, showing the command PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe c:/Users/Susmija/Desktop/AI/task2.py and its output 3.937007874015748. The right side of the interface includes a sidebar with icons for user context, task2.py, and AI tools like Ask and GPT-4.1.

```
# Input: 10
# Output: 3.937007874015748
print(cm_to_inches(10)) # Show
```

## Your Observations:

### Function Definition:

```
1.def cm_to_inches(cm):
```

- This defines a function that takes one parameter: cm (centimeters).
- A helpful docstring is provided, making the code easier to understand.

### Conversion Logic:

- return cm / 2.54
- Correct formula is used.  
( $1 \text{ inch} = 2.54 \text{ cm}$ )

### Function Call:

- print(cm\_to\_inches(10))
- Calls the function with 10 cm and prints the output.

### Output:

- 3.937007874015748

## TASK #3:

### Prompt:

- Few-shot: Provide 2–3 examples to generate a function that formats full names as “Last, First”.

### Code Generated:

The screenshot shows a code editor with several tabs open, including 'Student.py', 'task-2.py', 'def is\_leap\_year(year): Untitled-1', 'error.py', 'task2.py', 'task3.py', and 'CHAT'. The 'CHAT' tab contains a conversation with ChatGPT about creating a function to convert centimeters to inches. The code editor also displays examples of how the 'format\_name' function is used.

```

C: > Users > Susmija > Desktop > AI > task3.py > format_name
1 # Function to format full names as "Last, First"
2 def format_name(full_name):
3     """
4         Formats a full name as 'Last, First'.
5         Assumes the input is in 'First Last' format.
6     """
7     parts = full_name.strip().split()
8     if len(parts) < 2:
9         return full_name # Return as is if not enough parts
10    first, last = parts[0], parts[-1]
11    return f'{last}, {first}'
12
13 # Examples:
14 print(format_name("John Smith"))      # Output: Smith, John
15 print(format_name("Alice Johnson"))   # Output: Johnson, Alice
16 print(format_name("Bob Lee"))        # Output: Lee, Bob

```

## Output After executing Code:

The terminal window shows the command 'C:/Users/Susmija> & C:/Users/Susmija/anaconda3/python.exe c:/Users/Susmija/Desktop/AI/task3.py' being run. The output of the script is displayed, showing three formatted names: 'Smith, John', 'Johnson, Alice', and 'Lee, Bob'.

```

PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe c:/Users/Susmija/Desktop/AI/task3.py
Smith, John
Johnson, Alice
Lee, Bob
PS C:\Users\Susmija>

```

## Your Observations:

- The code defines a function `format_name(full_name)` that **formats a full name** from the form:
- "First Last" → "Last, First"

### 1.Function Definition:

```
def format_name(full_name):
```

- Takes a string input `full_name`.

### 2.Docstring and Assumptions:

Formats a full name as 'Last, First'!

Assumes the input is in 'First Last' format.

- Clearly states purpose and assumptions, which helps with code readability and maintenance.

### 3.Splitting the Name:

```
parts = full_name.strip().split()
```

- Removes extra spaces using `strip()`.
- Splits the name into parts (words) using `split()`.

### 4.Validation of Input:

```
if len(parts) < 2:
```

```
return full_name
```

- If there are not at least two words, it returns the input unchanged.

## 5.Name Reformatting:

1.first, last = parts[0], parts[-1]

2.return f"{last}, {first}"

- Assigns the **first** and **last** names and formats as "Last, First" using an f-string.
- Works even if there are middle names (e.g., "Alice Mary Johnson" → "Johnson, Alice")

## TASK #4:

### Prompt:

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

### Code Generated:

The screenshot shows the Visual Studio Code interface with several tabs open: task-2.py, def is\_leap\_year(year): Untitled-1, error.py, task2.py, task3.py, and task4.py. The task4.py tab is active, displaying the following Python code:

```
C: > Users > Susmija > Desktop > AI > task4.py > ...
1 def count_vowels(s):
2     return sum(1 for char in s.lower() if char in "aeiou")
3
4 # Example usage:
5 print(count_vowels("hello"))    # Output: 2
6 print(count_vowels("world"))   # Output: 1
7 print(count_vowels("AEIOU"))   # Output: 5
```

To the right of the code editor, there is an AI interface window titled "CHAT". It shows a zero-shot prompt: "counts the number of vowels in a string, give the python code". Below it, a few-shot prompt is shown: "Prompt: Write a function that counts the number of vowels in a string." and the generated code: "def count\_vowels(s): return sum(1 for char in s.lower() if char in 'aeiou')".

### Output After executing Code:

The screenshot shows the Visual Studio Code interface with the terminal tab selected. The terminal window displays the following output:

```
PS C:\Users\Susmija> & C:/Users/Susmija/anaconda3/python.exe c:/Users/Susmija/Desktop/AI/task4.py
● 2
1
5
PS C:\Users\Susmija>
```

To the right of the terminal, there is an AI interface window with a prompt: "Prompt: Given these examples, write a function that counts the number of vowels in a string." and the generated code: "# Example: count\_vowels("hello") # Example: count\_vowels("world") # Example: count\_vowels("AEIOU")".

### Your Observations:

-->The function `count_vowels(s)` counts how many **vowels** (a, e, i, o, u) are present in the input string `s`, **case-insensitively**.

### 1.Function Definition:

```
def count_vowels(s):
```

- Accepts a string inputs.

## 2.Core Logic:

return sum(1 for char in s.lower() if char in "aeiou")

- Converts the string to lowercase using `s.lower()` (ensures both upper and lower case are handled).
- Uses a **generator expression** inside `sum()` to:
  - Iterate over each character.
  - Count 1 for each character that is a vowel (a, e, i, o, u).
- Efficient, concise, and Pythonic implementation.

## 3.Test Cases and Output:

1. **Best Cases and Output:**
2. `print(count_vowels("hello")) # Output: 2`
3. `print(count_vowels("world")) # Output: 1`
4. `print(count_vowels("AEIOU")) # Output: 5`
  - The code prints the number of vowels in different test strings.
  - Terminal output confirms the expected results:
  - 2
  - 1
  - 5
5. **Handle Non-String Inputs (Edge Case Handling):**  
Add a type check:
6. `if not isinstance(s, str):`
7.  `raise TypeError("Input must be a string")`
8. **Return 0 for Empty Strings:**  
Already handled correctly (returns 0), but worth documenting.
9. **Consider Y as Vowel (If Required):**  
Some definitions include "y" as a vowel in certain contexts. If so:
10. `if char in "aeiouy"`
11. **Extended Test Cases (Optional):**
12. `print(count_vowels("Python 3.10"))`
13. `print(count_vowels(""))`
14. `print(count_vowels("sky"))`

## TASK #5:

### Prompt:

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

### Code Generated:

```

16 # Task:
17 # Write a function that reads a .txt file and returns the number of lines in it.
18
19 def count_lines_in_file(filename):
20     try:
21         with open(filename, 'r') as file:
22             lines = file.readlines()
23             return len(lines)
24     except FileNotFoundError:
25         return f"File '{filename}' not found."
26     except Exception as e:
27         return f"Error: {str(e)}"
28
29
30 # Example usage:
31 if __name__ == "__main__":
32     filename = input("Enter the file name (with .txt): ")
33     print("Number of lines:", count_lines_in_file(filename))

```

## Output After executing Code:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\musta\AppData\Local\Programs\Microsoft VS Code> & C:\Users\musta\AppData\Local\Programs\Python\Python313\python.exe c:/Users/musta/Desktop/Untitled-1.py
Enter the file name (with .txt): data.txt
Number of lines: File 'data.txt' not found.
PS C:\Users\musta\AppData\Local\Programs\Microsoft VS Code> & C:\Users\musta\AppData\Local\Programs\Python\Python313\python.exe c:/Users/musta/Desktop/Untitled-1.py
Enter the file name (with .txt): urls.txt
Number of lines: File 'urls.txt' not found.
PS C:\Users\musta\AppData\Local\Programs\Microsoft VS Code> & C:\Users\musta\AppData\Local\Programs\Python\Python313\python.exe c:/Users/musta/Desktop/Untitled-1.py
Enter the file name (with .txt): 

```

Spaces: 4 UTF-8 () Python Python 3.13 (64-bit)

## Your Observations

- The function count\_lines\_in\_file(filename) reads a .txt file and returns the number of lines.

### It includes error handling for:

- File not found
- Other unexpected exceptions
- 1. Function Definition and Logic:**

```
def count_lines_in_file(filename):
```

- Accepts the file name as a string input.
- with open(filename, 'r') as file:

```
    lines = file.readlines()
```

```
    return len(lines)
```

- Opens the file in read mode.
- Reads all lines into a list.
- Returns the number of lines using len().

## 2. Error Handling:

- except FileNotFoundError:

```
    return f"File '{filename}' not found."
```

- Specifically catches missing file errors and returns a friendly message.

except Exception as e:

- return f"Error: {str(e)}"
- Catches any other exceptions and prints the error message.

### 3. User Input and Example Usage:

```
if __name__ == "__main__":
```

- filename = input("Enter the file name (with .txt): ")
- print("Number of lines:", count\_lines\_in\_file(filename))

Takes filename input from the user.

Prints the result of count\_lines\_in\_file().

### 4. Terminal Output:

- Enter the file name (with .txt): data.txt
- Number of lines: File 'data.txt' not found.
- The file data.txt and urls.txt were not found in the directory, so the exception handling worked as expected.

l path; ensure it works correctly.

#### 1. Make It Case-Insensitive for Extension (Minor):

Allow .TXT, .Txt, etc., by checking lowercase extension:

```
2. if not filename.lower().endswith(".txt"):
```

```
    return "Please provide a valid .txt file"
```

- The code is **correct, robust, and user-friendly**.
- It effectively reads line counts from a file and handles missing files gracefully.
- Once a valid .txt file is present in the same directory, it will work perfectly.
- Would you like help creating a sample .txt file for testing?