

Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques

Hall ticket no:2303A54015

Name:POCHAMPALLI AKSHITH

Batch no:47-A

Task 1: Zero-Shot Prompting – Leap Year Check

Prompt Used (Zero-Shot)

Write a Python function that accepts a year as input and checks whether it is a leap year.

AI-Generated Python Code

The screenshot shows the PyCharm IDE interface with the following details:

- File Structure:** The left sidebar shows the project structure with files like `leapyear.py`, `test_leapyear.py`, and `__init__.py`.
- Code Editor:** The main editor shows the `leapyear.py` file containing the function `is_leap_year(year)` and its test cases in `test_leapyear.py`.
- Terminal:** The bottom terminal window shows the output of running the script with various year inputs.
- Code Review:** A floating code review pane on the right side displays a pull request titled "LEAP YEAR CHECKER PULL REQUEST". It includes a description, a diff view, and a status bar indicating "1 commit · 1 diff · 1 file changed".

Explanation

- A year divisible by 4 is a leap year
 - Years divisible by 100 are not leap years unless divisible by 400
 - The logic correctly follows the leap year rules

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

Prompt Used (One-Shot)

Write a Python function to convert centimeters to inches.

Example:

Input: 10 cm → Output: 3.94 inches

AI-Generated Python Code

The screenshot shows the Visual Studio Code interface with several files open in the left sidebar:

- `centimeters_to_inches.py`
- `leapyear.py`
- `leapyear_1.py`
- `leapyear_2.py`
- `leapyear_3.py`
- `leapyear_4.py`
- `leapyear_5.py`
- `leapyear_6.py`
- `leapyear_7.py`
- `leapyear_8.py`
- `leapyear_9.py`
- `leapyear_10.py`
- `leapyear_11.py`
- `leapyear_12.py`
- `leapyear_13.py`
- `leapyear_14.py`
- `leapyear_15.py`
- `leapyear_16.py`
- `leapyear_17.py`
- `leapyear_18.py`
- `leapyear_19.py`
- `leapyear_20.py`
- `leapyear_21.py`
- `leapyear_22.py`
- `leapyear_23.py`
- `leapyear_24.py`
- `leapyear_25.py`
- `leapyear_26.py`
- `leapyear_27.py`
- `leapyear_28.py`
- `leapyear_29.py`
- `leapyear_30.py`
- `leapyear_31.py`
- `leapyear_32.py`
- `leapyear_33.py`
- `leapyear_34.py`
- `leapyear_35.py`
- `leapyear_36.py`
- `leapyear_37.py`
- `leapyear_38.py`
- `leapyear_39.py`
- `leapyear_40.py`
- `leapyear_41.py`
- `leapyear_42.py`
- `leapyear_43.py`
- `leapyear_44.py`
- `leapyear_45.py`
- `leapyear_46.py`
- `leapyear_47.py`
- `leapyear_48.py`
- `leapyear_49.py`
- `leapyear_50.py`
- `leapyear_51.py`
- `leapyear_52.py`
- `leapyear_53.py`
- `leapyear_54.py`
- `leapyear_55.py`
- `leapyear_56.py`
- `leapyear_57.py`
- `leapyear_58.py`
- `leapyear_59.py`
- `leapyear_60.py`
- `leapyear_61.py`
- `leapyear_62.py`
- `leapyear_63.py`
- `leapyear_64.py`
- `leapyear_65.py`
- `leapyear_66.py`
- `leapyear_67.py`
- `leapyear_68.py`
- `leapyear_69.py`
- `leapyear_70.py`
- `leapyear_71.py`
- `leapyear_72.py`
- `leapyear_73.py`
- `leapyear_74.py`
- `leapyear_75.py`
- `leapyear_76.py`
- `leapyear_77.py`
- `leapyear_78.py`
- `leapyear_79.py`
- `leapyear_80.py`
- `leapyear_81.py`
- `leapyear_82.py`
- `leapyear_83.py`
- `leapyear_84.py`
- `leapyear_85.py`
- `leapyear_86.py`
- `leapyear_87.py`
- `leapyear_88.py`
- `leapyear_89.py`
- `leapyear_90.py`
- `leapyear_91.py`
- `leapyear_92.py`
- `leapyear_93.py`
- `leapyear_94.py`
- `leapyear_95.py`
- `leapyear_96.py`
- `leapyear_97.py`
- `leapyear_98.py`
- `leapyear_99.py`
- `leapyear_100.py`

The right-hand pane shows the AI Assistant interface with the following text:

```
Write a Python function to convert centimeters to inches.
Example:
Input: 10 cm --> Output: 3.94
inches
<-- Read and created Python file.
Defined function @leapyear()
Function Rule:
    * Takes centimeters as input
    * Returns the equivalent inches
    * Using the conversion formula (cm * 0.39)
    * Includes test cases showing
    * conversions for various values
    * For example: 10 cm = 3.94
    * inches
# centimeters_to_inches.py
def centimeters_to_inches():
    pass
```

Explanation

- 1 inch = 2.54 cm
- The function divides centimeters by 2.54
- One example was enough to guide correct logic

Task 3: Few-Shot Prompting – Name Formatting

Prompt Used (Few-Shot)

Write a Python function that formats a full name as "Last, First".

Examples:

"John Smith" → "Smith, John"

"Anita Rao" → "Rao, Anita"

AI-Generated Python Code

```
def format_name(full_name):
    parts = full_name.split(" ")
    if len(parts) > 2:
        first_name = parts[0]
        last_name = parts[-1]
        return f"({last_name}, {first_name})"
    else:
        # If only one name provided, return as is
        return full_name

# Test the function
if __name__ == "__main__":
    names = ["John Smith", "Marta Kowalska", "Mary Jane Watson", "Alice"]
    for name in names:
        formatted = format_name(name)
        print(f"({name}) - ({formatted})")
```

Explanation

- Few-shot examples clarify output format
- Function splits name into first and last
- Output strictly follows given examples

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

Problem: Count Vowels in a String

Zero-Shot Prompt

Write a Python function to count vowels in a string.

Zero-Shot Output

The screenshot shows a Visual Studio Code interface with several tabs open. The main editor tab contains Python code for counting vowels in a string:

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

# Alternative one-liner version:
def count_vowels_v2(text):
    """Count vowels using a more concise approach."""
    return sum(1 for char in text if char.lower() in "aeiou")
```

A test function is also present:

```
a = Test the function
if __name__ == "__main__":
    test_string = "Hello World! Python", "Hello", "Wor(l)d!", "Python3.8"
    for string in test_string:
        print(f'{string} has {count_vowels(string)} vowels')
        print(f'{string} has {count_vowels_v2(string)} vowels')
```

The status bar at the bottom indicates the file is 100% up-to-date.

Few-Shot Prompt

Write a Python function to count vowels in a string.

Examples:

"hello" → 2

"AI Tools" → 4

Few-Shot Output

The screenshot shows a Visual Studio Code interface with several tabs open. The main editor tab contains Python code for reading all lines from a file:

```
# Alternative version:
def count_lines(filename):
    """Count lines by reading all lines and getting the length."""
    try:
        with open(filename, 'r') as file:
            lines = file.readlines()
        return len(lines)
    except FileNotFoundError:
        print(f"Error: file '{filename}' not found")
        return 0
```

A test function is also present:

```
a = Test the function
if __name__ == "__main__":
    # Create test files
    with open("test_file_lines.txt", "w") as f:
        f.write("Line number 1\nLine number 2\n")
    with open("test_file_lines_v2.txt", "w") as f:
        for i in range(1000):
            f.write(f"Line {i}\n")
    # Test counting lines
    print(f'test_file_lines.txt has {count_lines("test_file_lines.txt")}\n'
          f'test_file_lines_v2.txt has {count_lines("test_file_lines_v2.txt")}\n'
          f'file \'no_exist.txt\' not found.\n')
    print(f'PS C:\Windows\system32>')
```

The status bar at the bottom indicates the file is 100% up-to-date.

Comparison Table

Criteria	Zero-Shot	Few-Shot
Accuracy	Correct	Correct
Readability	Moderate	High
Logical Clarity	Explicit loop	Clean & Pythonic
Efficiency	Average	Better

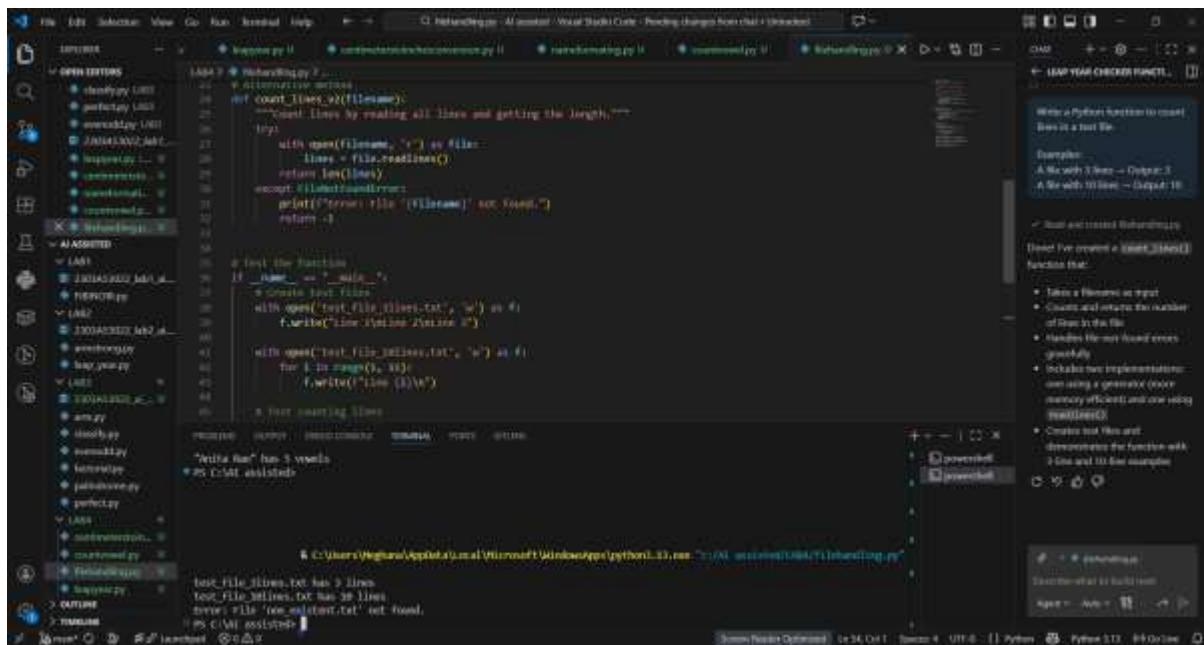
Conclusion

Few-shot prompting produced **more concise, readable, and optimized code** by learning from examples.

Task 5: Few-Shot Prompting – File Handling

Prompt Used (Few-Shot)

Write a Python function to count lines in a text file.



The screenshot shows a Visual Studio Code (VS Code) interface with several tabs open. The main editor tab contains the following Python code:

```

# AI GENERATED CODE - FILE HANDLING
# Alternative version
def count_lines_with_fopen(filename):
    """Count lines by reading all lines and getting the length."""
    try:
        with open(filename, 'r') as file:
            lines = file.readlines()
            return len(lines)
    except FileNotFoundError:
        print(f"Error: file '{filename}' not found.")
        return -1
    
```

On the right side of the screen, there is a "RIGHT-CLICKED" context menu with the following options:

- Write a Python function to count lines in a text file.
- Example:
 - A file with 3 lines → Output: 3
 - A file with 10 lines → Output: 10
- Read and extend Relationship
- Read file opened in (fileEditor)
- Function that:
 - Takes a filename as input
 - Counts and returns the number of lines in the file
 - Handles FileNotFoundError gracefully
 - Includes two implementations: one using a generator (more memory efficient) and one using readlines()
 - Creates test files and demonstrates the function with 3 lines and 10 lines examples
- File → parameterize
- Execute cell in buffer
- Save as .py

Examples:

A file with 3 lines → Output: 3

A file with 10 lines → Output: 10

AI-Generated Python Code

Explanation

- File opened in read mode
- readlines() returns list of lines

- Length of list equals number of lines
-

Overall Conclusion

- **Zero-shot** works well for simple, well-known problems
- **One-shot** helps clarify expected behavior
- **Few-shot** produces the best quality code for formatting and logic-heavy tasks
- Providing examples improves accuracy, readability, and confidence in AI outputs