

Hall ticket no:2303A54006

Batch no:47-A

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

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 a Visual Studio Code interface with the following details:

- File Explorer:** Shows files like `leapyear.py`, `is_leap_year.py`, and `test_leapyear.py`.
- Code Editor:** Displays the `leapyear.py` file containing:

```
def is_leap_year(year):
    """Check if the year is a leap year. Return True or False.

    If (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

# Test the function
if __name__ == "__main__":
    test_years = [2000, 2004, 2005, 2006, 2008]
    for year in test_years:
        if is_leap_year(year):
            print(f"{year} is a leap year!")
        else:
            print(f"{year} is not a leap year!")
```
- Terminal:** Shows the output of running the script:

```
PS C:\AI assistants & ChatGPT\chatgpt\src\Visual Studio Code\Python\leapyear> python leapyear.py
2004 is a leap year!
2005 is not a leap year!
2006 is a leap year!
2008 is not a leap year!
2000 is a leap year!
PS C:\AI assistants & ChatGPT\chatgpt\src\Visual Studio Code\Python\leapyear>
```
- Right Panel:** Shows the AI's thought process and context for generating the code.

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 main editor window contains the following Python code:

```
def cm_to_inches(cm):
    inches = cm / 2.54
    return inches

def test():
    test_values = [10, 25, 50, 100, 1]
    for cm in test_values:
        result = cm_to_inches(cm)
        print(f"{cm} cm is {result:.2f} inches")

if __name__ == "__main__":
    test()
```

The status bar at the bottom indicates "Syntax Reader Optimized" and "Python 3.13 - Inactive".

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
```

The screenshot shows a Visual Studio Code interface. In the center, there's a code editor with the following 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
```

To the right of the code editor, a completion assistant window is open. It has a title "LEAP YEAR CHECKER FUNCTION". Inside, it says "Format a full name as '(last, first)'". Below that, under "Examples", it shows two examples: "John Smith" → "(Smith, John)" and "Jane Doe" → "(Doe, Jane)". At the bottom of the completion window, there's a list of "Function hints":

- Takes a full name as input
- Splits it and rearranges to "(last, first)" format
- Handles edge cases (single names are returned as is)
- Includes test cases:
 - John Smith → "(Smith, John)"
 - Jane Doe → "(Doe, Jane)"

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

```

1 #!/usr/bin/python3
2 def count_vowels(text):
3     vowels = "aeiouAEIOU"
4     count = 0
5
6     for char in text:
7         if char in vowels:
8             count += 1
9
10    return count
11
12 # Alternative one-liner version:
13 def count_vowels_v2(text):
14     """Count vowels using a more concise approach."""
15     return sum(1 for char in text if char.lower() in "aeiou")
16
17 # Test the function:
18 if __name__ == "__main__":
19     test_strings = ["Hello world!", "Python", "aiML", "xyz", "Hello Kode"]
20
21     for string in test_strings:
22         print(f'{string} has {count_vowels(string)} vowels')
23
24         <-- c:\users\wagners\appdata\local\microsoft\windowsapps\python3.8\python.exe "c:/ml/assistant/1044/countvowels.py"
25
26         "Hello world!" has 3 vowels
27         "Python" has 1 vowels
28         "aiML" has 3 vowels
29         "xyz" has 0 vowels
30         "Hello Kode" has 5 vowels
31
32 PS C:\ml\assistant>

```

Few-Shot Prompt

Write a Python function to count vowels in a string.

Examples:

"hello" → 2

"AI Tools" → 4

Few-Shot Output

```

1 #!/usr/bin/python3
2 def count_lines(filename):
3     """Count lines by reading all lines and getting the length."""
4     try:
5         with open(filename, 'r') as file:
6             lines = file.readlines()
7             return len(lines)
8     except FileNotFoundError:
9         print(f"Error: file '{filename}' not found")
10        return 0
11
12 # Test the function:
13 if __name__ == "__main__":
14     # Create test files:
15     with open("test_file_lines.txt", "w") as f:
16         f.write("Line number 1\nLine number 2\nLine number 3")
17
18     with open("test_file_vowels.txt", "w") as f:
19         for i in range(0, 10):
20             f.write(f"Line {i}\n")
21
22     # Test counting lines:
23
24         <-- c:\users\wagners\appdata\local\microsoft\windowsapps\python3.8\python.exe "c:/ml/assistant/1044/readlines.py"
25
26         test_file_lines.txt has 3 lines
27         test_file_vowels.txt has 10 lines
28         Error: file 'no_such_file.txt' not found.
29
30 PS C:\ml\assistant>

```

Comparison Table

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

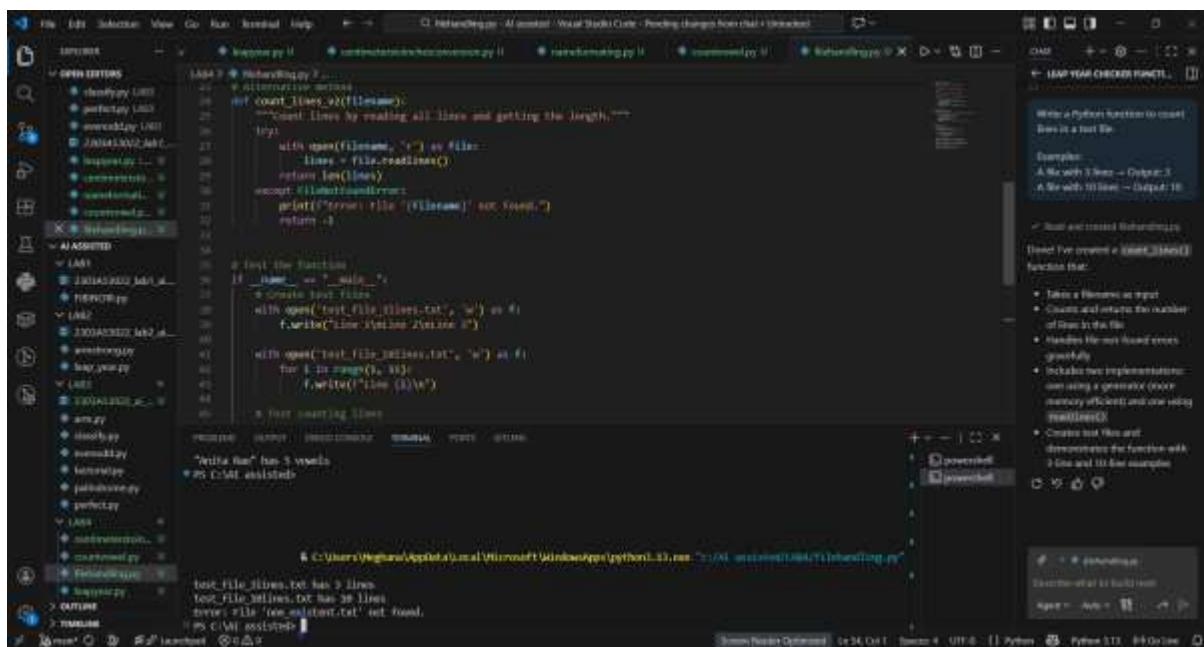
Conclusion

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 "HAPPY CODER PORTAL" sidebar with the following text:

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

Below the example, it says:

Result and created Relationship

Result: File opened in `read mode`

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

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