Name: Padmaa.k.b
SRN: PES2UG23CS254
Section: D
1st one :

[13]:
```python
from langchain_core.prompts import ChatPromptTemplate

# Create template
template = ChatPromptTemplate.from_template(
    "Given the movie {movie}, tell its release year and how many years ago it was released."
)

# Create chain using LCEL
chain = template | llm | parser

# Run the chain
result = chain.invoke({"movie": "Inception"})

print(result)
```

```
The movie Inception was released in **2010**.

As of 2024, that means it was released **14 years ago**.
```

2nd one :

[9]:
```python
structured_prompt = """
# Context
You are a Senior Python Developer.

# Objective
Write a Python function to reverse a string.

# Constraints
1. Use recursion only.
2. Do NOT use slicing (no [::-1]).
3. Do not use built-in reverse functions.

# Style
Write clean and readable code.
Include detailed docstrings explaining the function, parameters, and return value.

# Audience
Beginner Python programmers.

# Output Format
Only provide the Python function code.
"""

print(llm.invoke(structured_prompt).content)
```

```
```python
```
```

```python
def reverse_string_recursive(s: str) -> str:
    """
    Reverses a given string using recursion.

    This function takes an input string and returns a new string with its
    characters arranged in reverse order. It achieves this by defining a
    base case for very short strings and a recursive step that breaks down
    the problem into smaller, similar subproblems.

    Constraints followed:
    1.  Uses recursion exclusively; no explicit loops (for, while) are used.
    2.  Does NOT use string slicing for reversal (e.g., `s[::-1]`).
    3.  Does NOT use built-in reverse functions (e.g., `reversed()`).

    Args:
        s (str): The input string that needs to be reversed.

    Returns:
        str: A new string containing the characters of the input string
             in reverse order.

    Examples:
        >>> reverse_string_recursive("hello")
        "olleh"
        >>> reverse_string_recursive("Python")
        "nohtyP"
        >>> reverse_string_recursive("")
        ""
        >>> reverse_string_recursive("a")
        "a"
        >>> reverse_string_recursive("racecar")
        "racecar"
    """
    # Base Case:
    # If the string is empty or contains only one character, it is
    # already considered "reversed". This is the stopping condition
    # for our recursion, preventing an infinite loop.
    if len(s) <= 1:
        return s
    else:
        # Recursive Step:
        # 1. Get the first character of the string.
        first_char = s[0]
        # 2. Get the rest of the string, which includes all characters
        #    from the second character up to the end.
        #    This is NOT slicing for reversal, but standard string indexing
        #    to get a substring starting from index 1.
        rest_of_string = s[1:]
        # 3. Recursively call the function on the 'rest_of_string'.
        #    This means we're asking the function to reverse a smaller part
        #    of the original string.
        reversed_rest = reverse_string_recursive(rest_of_string)
        # 4. Combine the results: Append the 'first_char' to the end of
        #    the 'reversed_rest'. This action effectively moves the first
        #    character of the original string to the very last position
        #    in the final reversed string.
        return reversed_rest + first_char
```

3. Analysis

Using `FewShotChatMessagePromptTemplate` creates a clean separation between instructions and data. This helps the Attention Mechanism focus on the right things.

[ ]:
```
FewShotChatMessagePromptTemplate separates instructions from examples.
This makes the prompt more structured and easier for the model to understand.

The Attention Mechanism focuses on the examples to learn the pattern
and applies it to generate better and more accurate outputs.
```

[ ]:

3rd one :

[ ]:
```
Chain of Thought prompting encourages the model to think step by step.
This improves reasoning and gives more accurate answers.
```

[ ]:
```
Tree of Thoughts explores multiple solutions and selects the best one.
It improves decision making.
```

[ ]:
```
Graph of Thoughts combines multiple ideas into a final output.
It is useful for creative and complex tasks.
```