

Assignment-4.3

Name: Y. Poojitha

Hall ticket No:2303A51499

Batch:08

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

Task 1: Zero-Shot Prompting – Leap Year Check

The screenshot shows the VS Code interface with the file `10_2.py` open. The code defines a function `is_leap_year` that checks if a year is a leap year based on the rules: divisible by 400, or divisible by 100 but not by 400, or divisible by 4 but not by 100. It also includes a test case loop that prints the status of each year in the list [2000, 1900, 2004, 2001, 2020, 2021, 2024, 1996, 2100, 2400]. The sidebar on the right contains a "CHAT" section with a message from the AI assistant, a "PALINDROME CHECKER" section, and a "Leap Year Rules:" section with the following items:

- Divisible by 400
- Divisible by 100 but not a leap year (e.g., 1900)
- Divisible by 4 but not by 100 (e.g., 2004)
- Otherwise → Not a leap year

Test cases include:

- 2000, 2004, 2020
- 1900, 2001, 2021

Output:

The screenshot shows the VS Code terminal tab with the command `python 10_2.py` run. The output shows the program's response to each year in the test list:

```
PS D:\AI_assistant_coding> & 'c:\Users\yarav\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\yarav\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61916' '--' 'd:\AI_assistant_coding\10_2.py'
2000 → Leap Year
1900 → Not a Leap Year
2004 → Leap Year
2001 → Not a Leap Year
2020 → Leap Year
2021 → Not a Leap Year
2024 → Leap Year
1996 → Leap Year
2100 → Not a Leap Year
2400 → Leap Year
PS D:\AI_assistant_coding>
```

Task 2: One-Shot Prompting

Centimeters to Inches Conversion

The screenshot shows the VS Code interface with the following details:

- Editor:** The code editor displays a Python file named `10_2.py`. It contains a function `cm_to_inches` and a series of test cases for it.
- Terminal:** The terminal window shows the output of running the script, displaying various centimeter-to-inch conversions.
- Output:** The output panel shows a summary of the task completed.
- Debug Console:** The Python Debug Console shows the command used to run the script.
- Chat:** A sidebar titled "PALINDROME CHECK FUNCTION IN PYT..." is visible.
- Task Bar:** A task bar at the top indicates "Task 2: One-Shot Prompting - Centimeters to Inches Conversion".
- Key Features:**
 - Uses the standard conversion: 1 inch = 2.54 cm
 - Takes centimeters as input and returns inches
 - Rounds the result to 2 decimal places for accurate measurements
 - Includes test cases demonstrating the conversion
- Test Cases:**
 - 10 cm → 3.94 inches
 - 25 cm → 9.84 inches
 - 50 cm → 19.69 inches
 - 100 cm → 39.37 inches
 - 154 cm → 60.63 inches
 - 5.5 cm → 2.17 inches
 - 30.48 cm → 12.0 inches

Task 3: Few-Shot Prompting – Name Formatting

The screenshot shows the VS Code interface with the following details:

- Editor:** The code editor displays a Python file named `10_2.py`. It contains a function `format_name` and a series of test cases for it.
- Terminal:** The terminal window shows the output of running the script, displaying various name-formatting examples.
- Output:** The output panel shows a summary of the task completed.
- Debug Console:** The Python Debug Console shows the command used to run the script.
- Chat:** A sidebar titled "PALINDROME CHECK FUNCTION IN PYT..." is visible.
- Task Bar:** A task bar at the top indicates "Task 2: One-Shot Prompting - Centimeters to Inches Conversion".
- Key Features:**
 - Takes first and last names as separate inputs
 - Converts names to title case (first letter capitalized)
 - Handles mixed input cases (lowercase, uppercase, or mixed)
 - Returns a properly formatted full name
- Few-Shot Examples Included:**
 - john, doe → John Doe
 - jane, smith → Jane Smith
 - MARY, JOHNSON → Mary Johnson
 - robert, brown → Robert Brown
 - alice, williams → Alice Williams
 - michael, jones → Michael Jones
- Description:** A note states: "This demonstrates the few-shot prompting pattern with multiple examples showing the expected input-output format."

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

```

File Edit Selection View Go Run Terminal Help < > | AI_assistant_coding
OPEN EDITORS Welcome 10.2.py ...
EXPLORER
10.2.py > ...
1 def classify_sentiment_zero_shot(text):
2     return "Zero-Shot Classification"
3
4 def classify_sentiment_few_shot(text):
5     return "Few-Shot Classification"
6
7 if __name__ == "__main__":
8     print("=" * 70)
9     print("ZERO-SHOT vs FEW-SHOT PROMPTING COMPARISON")
10    print("=" * 70)
11    print()
12
13    # Zero-Shot approach
14    print("ZERO-SHOT APPROACH:")
15    print("=" * 70)
16    print("\u2248 No examples provided")
17    print("\u2248 Direct instruction only")
18    print("\u2248 Requires more general knowledge from the model")
19    print("\u2248 Faster to set up")
20    print("\u2248 May produce less consistent results")
21
22    result_zero = classify_sentiment_zero_shot("This is amazing!")
23    print(f"Result: {result_zero}")
24    print()
25
# Few-Shot approach
print("FEW-SHOT APPROACH:")
print("-" * 70)
print("\u2248 Multiple examples provided (typically 2-5)")
print("\u2248 Demonstrates the expected pattern")
print("\u2248 Guides the model through similar examples")
print("\u2248 Produces more consistent and accurate results")
print("\u2248 Requires more tokens/computation")
print()
result_few = classify_sentiment_few_shot("This is amazing!")
print(f"Result: {result_few}")
print()

# Key Differences
print("KEY DIFFERENCES:")
print("-" * 70)
comparison_data = [
    ("Examples", "None", "2-5+ examples"),
    ("Consistency", "Lower", "Higher"),
    ("Accuracy", "Lower", "Higher"),
    ("Setup Time", "Faster", "Slower"),
    ("Token Usage", "Lower", "Higher"),
    ("Use Case", "Simple tasks", "Complex/nuanced tasks"),
]

```

Zero-Shot Prompting:

- No examples provided, just the instruction
- Direct approach
- Faster to set up but less consistent
- Relies on model's general knowledge

Few-Shot Prompting:

- Provides examples demonstrating the pattern
- Guides the model through similar cases
- More consistent and accurate results
- Uses more tokens/computation

```

# Few-Shot approach
print("FEW-SHOT APPROACH:")
print("-" * 70)
print("\u2248 Multiple examples provided (typically 2-5)")
print("\u2248 Demonstrates the expected pattern")
print("\u2248 Guides the model through similar examples")
print("\u2248 Produces more consistent and accurate results")
print("\u2248 Requires more tokens/computation")
print()
result_few = classify_sentiment_few_shot("This is amazing!")
print(f"Result: {result_few}")
print()

# Key Differences
print("KEY DIFFERENCES:")
print("-" * 70)
comparison_data = [
    ("Examples", "None", "2-5+ examples"),
    ("Consistency", "Lower", "Higher"),
    ("Accuracy", "Lower", "Higher"),
    ("Setup Time", "Faster", "Slower"),
    ("Token Usage", "Lower", "Higher"),
    ("Use Case", "Simple tasks", "Complex/nuanced tasks"),
]

for feature, zero, few in comparison_data:
    print(f"{feature:15} | Zero-Shot: {zero:20} | Few-shot: {few:20}")

```

The file includes implementation details and would structure the comparison data.