AIAssistedCodig

Assignment - 03

Name : K.Srija

RollNo : 2303A54023

Batch – 48

Lab Experiment: Prompt Engineering – Improving Prompts and Context Management

Task1: Write a **zero-shot prompt** (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

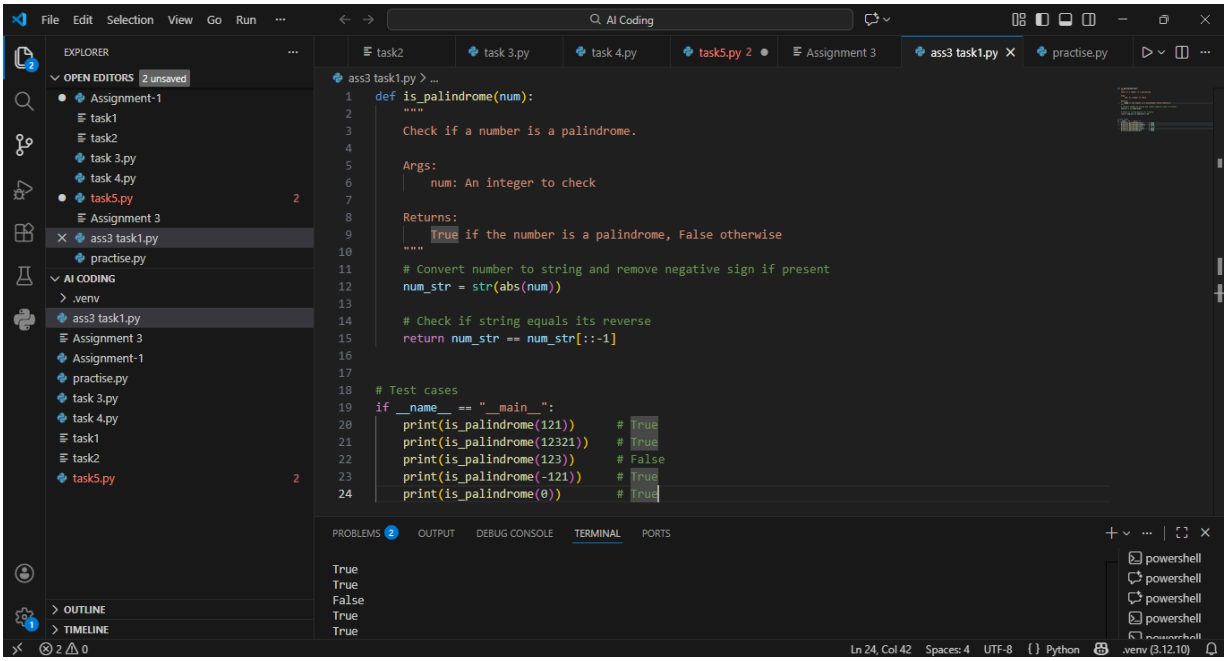
Scenario: You are building a **basic number-validation utility** for a digital system where users enter numeric IDs.
The system must automatically check whether a given number is a **palindrome** .You want to evaluate how well an AI can generate correct Python logic **without any prior examples**, relying only on the task description

- **Prompt used:**
  "Generate a Python function that checks whether a given number is a palindrome. The function should return True if the number is a palindrome and False otherwise. Use basic programming constructs only."

- **Generatedcode:**

```
def is_palindrome(num):
    """
    Check if a number is a palindrome.

    Args:
        num: An integer to check

    Returns:
        True if the number is a palindrome, False otherwise
    """
    # Convert number to string and remove negative sign if present
    num_str = str(abs(num))

    # Check if string equals its reverse
    return num_str == num_str[::-1]


# Test cases
if __name__ == "__main__":
    print(is_palindrome(121))    # True
    print(is_palindrome(12321))  # True
```

```
print(is_palindrome(123))     # False
print(is_palindrome(-121))    # True
print(is_palindrome(0))       # True
```



- **Short Explanation of Logic:**

In this scenario, zero-shot prompting means asking the AI to write a Python function **without giving any examples**. The AI is only told to check whether a given number is a palindrome or not.Based on the prompt, the AI generates a function that reverses the number and compares it with the original number. If both are the same, the number is a palindrome.The generated code is then tested using different inputs like palindrome numbers (121), non-palindrome numbers (123), and other values. Testing helps to check whether the function works correctly.While testing, some problems may be found. The code may not handle **zero or negative numbers properly** because these cases were not mentioned in the prompt. This shows that zero-shot prompting can miss special cases.

Task2: : One-Shot Prompting (Factorial Calculation)

Scenario: Write a **one-shot prompt** by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

- **Prompt used:**
- **"Example: Input: 5 → Output: 120. Using this example, write a Python function to calculate the factorial of a given number."**
- **Generatedcode:**

```
def factorial(n):
    """
    Calculate the factorial of a given number.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)


# Example usage
print(factorial(5))  # Output: 120
```

- **Sample Input:**
- Input: 5
- **SampleOutput:**
- 120
- **Short Explanation of Logic:**
- In one-shot prompting, the AI is given **one example** (5 → 120) along with the task. This example helps the AI clearly understand how factorial works. Compared to zero-shot, the generated code is more accurate and clear, especially in handling loops and return values. The example reduces confusion and improves correctness.

Task3: Few-Shot Prompting (Armstrong Number Check)

Scenario: Write a **few-shot prompt** by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

- **Prompt used:**
  "Input: 153 → Output: Armstrong Number Input: 370 → Output: Armstrong Number Input: 123 → Output: Not an Armstrong Number Using these examples, write a Python function to check whether a number is an Armstrong number."
- **Generatedcode:**

```
def is_armstrong_number(num):
    # Convert the number to string to iterate over digits
    digits = str(num)
    power = len(digits)

    # Calculate the sum of each digit raised to the power of the number of digits
    armstrong_sum = sum(int(digit) ** power for digit in digits)

    # Check if the sum is equal to the original number
    return armstrong_sum == num

# Test the function with examples
test_numbers = [153, 370, 123]
for number in test_numbers:
    if is_armstrong_number(number):
        print(f"Input: {number} → Output: Armstrong Number")
    else:
        print(f"Input: {number} → Output: Not an Armstrong Number")
```

- **SampleInput:**

- Input: 153 → Output: Armstrong Number
- Input: 370 → Output: Armstrong Number
- Input: 123 → Output: Not an Armstrong Number

SHORT LOGIC EXPLANATION:

In few-shot prompting, **multiple examples** are provided. These examples help the AI identify the exact pattern needed to check Armstrong numbers. Because of multiple inputs, the AI generates better-structured code and correctly applies the power logic. Testing with different values shows higher accuracy and fewer logical errors.

"""

Task4: Context-Managed Prompting (Optimized Number Classification)

Scenario: Design a **context-managed prompt** with clear instructions and constraints to generate an optimized Python program that classifies a number as **prime, composite, or neither**.

Prompt used:

"Write an optimized Python program that classifies a number as prime, composite, or neither. Validate input and ensure efficient logic."

- **GENERATE CODE:**

```
def classify_number(num):
    if not isinstance(num, int):
        return "Input must be an integer."
    if num < 0:
        return "Neither prime nor composite."
    if num in (0, 1):
        return "Neither prime nor composite."

    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return "Composite"
    return "Prime"

# Input validation
try:
    number = int(input("Enter an integer: "))
    result = classify_number(number)
    print(f"The number {number} is classified as: {result}")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

EXPLANATION:

In context-managed prompting, detailed instructions and constraints are given. This helps the AI generate optimized code with proper input validation. The logic becomes more efficient by reducing unnecessary checks. Compared to earlier prompts, the output is more reliable and handles edge cases correctly.

Task5: Zero-Shot Prompting (Perfect Number Check)

Scenario: Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

- **Promptused: "Generate a Python function to check whether a given number is a perfect number.**

Generatedcode:

```
def is_perfect_number(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n

# Example usage
```

```
number = 28
if is_perfect_number(number):
    print(f"{number} is a perfect number.")
else:
    print(f"{number} is not a perfect number.")
```

LOGIC EXPLANATION :

In zero-shot prompting, no examples are given. The AI generates basic logic to find divisors and check if their sum equals the number. However, some conditions like negative numbers or zero may not be handled properly. Testing helps identify missing checks and inefficiencies.

Task6: Few-Shot Prompting (Even or Odd Classification with Validation)

Scenario: Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Prompt used:

"Input: 8 → Output: Even Input: 15 → Output: Odd Input: 0 → Output: Even Write a Python program to determine whether a number is even or odd with proper input validation."

- **GENERATE CODE:**

```
def is_even_or_odd(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

def main():
    try:
        user_input = int(input("Enter a number: "))
        result = is_even_or_odd(user_input)
        print(f"Input: {user_input} → Output: {result}")
    except ValueError:
        print("Invalid input! Please enter an integer.")

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

. SAMPLE INPUT :

- Input: 8
- Input: 15
- Input: 0

SAMPLE OUTPUT :

- Output: Even
- Output: Odd

- Output: Even

LOGIC EXPLANATION :

Providing multiple examples helps the AI clearly understand even and odd classification. The generated program includes better input validation and clearer output messages. Testing with negative and non-integer inputs shows improved handling compared to zero-shot prompting.