Name: Achi Vaishnavi

Hall cket:2303A51554

Batch:29

Assessment: 3.1

Task 1: Zero-Shot Prompting (Palindrome Number Program)

PROMPT :

Generate a Python function that checks whether a given number is a palindrome.

CODE and OUTPUT :

```python
def is_number_palindrome(number):
    """
    Check if a given number is a palindrome.

    A number is a palindrome if it reads the same forwards and backwards.

    Parameters:
    number (int): The number to check.

    Returns:
    bool: True if the number is a palindrome, False otherwise.
    """
    # Convert the number to string to easily reverse it
    str_number = str(number)

    # Compare the string with its reverse
    return str_number == str_number[::-1]
    cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
    return cleaned_s == cleaned_s[::-1]
# Example Usage:
print(is_number_palindrome(121))  # True
print(is_number_palindrome(-121)) # False
print(is_number_palindrome(10))   # False
print(is_number_palindrome(12321)) # True
print(is_number_palindrome(1234321)) # True
```

```
True
False
False
True
True
```

Justification :

This task is significant because it evaluates the AI's basic understanding when only a problem statement is provided. The AI can generate a working palindrome-checking solution using general knowledge. However, the absence of examples and constraints often leads to missing edge-case handling, such as negative or invalid inputs. This shows that zero-shot prompting is suitable for simple tasks but may not produce fully reliable, real-world-ready code.

Task 2: One-Shot Prompting (Factorial Calculation)

PROMPT :

Generate a Python function to compute the factorial of a given number.

Example: Input: 5 → Output: 120.

CODE and OUTPUT :

```python
def compute_factorial(n):
    """
    Compute the factorial of a given number.

    The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.

    Parameters:
    n (int): The number to compute the factorial for.

    Returns:
    int: The factorial of the number.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    elif n == 0 or n == 1:
        return 1
    else:
        factorial = 1
        for i in range(2, n + 1):
            factorial *= i
        return factorial
# Example Usage:
print(compute_factorial(5))  # Output: 120
print(compute_factorial(0))  # Output: 1
print(compute_factorial(1))  # Output: 1
print(compute_factorial(7))  # Output: 5040
```

```
120
1
1
5040
```

Justification :

This task is significant because the single example clearly defines the expected behavior of the factorial function. The example helps the AI structure the logic more accurately and reduces ambiguity. Compared to zero-shot prompting, the generated code shows improved correctness and clearer implementation. This experiment demonstrates that even one reference example can greatly enhance the quality and reliability of AI-generated programs.

Task 3: Few-Shot Prompting (Armstrong Number Check)

PROMPT :

Generating a Python function to check whether a given number is an Armstrong number.

Examples:

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

CODE and OUTPUT :

```python
def is_armstrong_number(number):
    """
    Check if a given number is an Armstrong number.
    An Armstrong number (or narcissistic number) is a number that is equal to the sum of its own digits each raised
    Parameters:
    number (int): The number to check.
    Returns:
    bool: True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to string to easily iterate over digits
    str_number = str(number)
    num_digits = len(str_number)
    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_number)

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == number
# Example Usage:
print(is_armstrong_number(153))  # Output: True
print(is_armstrong_number(370))  # Output: True
print(is_armstrong_number(123))  # Output: False
print(is_armstrong_number(9474)) # Output: True
print(is_armstrong_number(9475)) # Output: False
```

```
True
True
False
True
False
```

Justification :

This task is significant because providing multiple examples clearly trains the AI to recognize the Armstrong number pattern. The examples guide the AI in correctly processing digits, applying powers, and making decisions. As a result, the generated code is more structured, accurate, and aligned with mathematical definitions. Boundary and invalid input testing further shows better generalization, proving that few-shot prompting leads to more reliable and practical solutions.

Task 4: Context-Managed Prompting (Optimized Number Classification)

PROMPT :

Generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

CODE and OUTPUT :

```python
def classify_number(num):
    """Classify a number as prime, composite, or neither.
    Parameters:
    num (int): The number to classify.
    Returns:
    str: "Prime", "Composite", or "Neither"."""
    # Input validation
    if not isinstance(num, int):
        return "Input must be an integer."
    if num < 0:
        return "Neither"
    if num == 0 or num == 1:
        return "Neither"
    # Check for primality
    if num == 2:
        return "Prime"
    if num % 2 == 0:
        return "Composite"

    for i in range(3, int(num**0.5) + 1, 2):
        if num % i == 0:
            return "Composite"
    return "Prime"
# Example Usage:
print(classify_number(11))  # Output: Prime
print(classify_number(15))  # Output: Composite
print(classify_number(1))   # Output: Neither
print(classify_number(0))   # Output: Neither
```

```
Prime
Composite
Neither
Neither
```

Justification :

This task is significant because context-managed prompting directs the AI with clear goals, rules, and constraints. By defining validation and optimization requirements, the AI generates a more efficient and complete solution. The program correctly handles edge cases such as 0, 1, and negative numbers, and applies optimized logic for classification. Compared to earlier prompting methods, the output is more structured and professional, proving that context and constraints are crucial for complex tasks.

Task 5: Zero-Shot Prompting (Perfect Number Check)

PROMPT :

Generate a Python function that checks whether a given number is a perfect number.

CODE and OUTPUT :

```python
def is_perfect_number(number):
    """
    Check if a given number is a perfect number.

    A perfect number is a positive integer that is equal to the sum of its proper positive divisors, excluding itsel

    Parameters:
    number (int): The number to check.

    Returns:
    bool: True if the number is a perfect number, False otherwise.
    """
    if number <= 0:
        return False

    # Calculate the sum of proper divisors
    sum_of_divisors = 0
    for i in range(1, number // 2 + 1):
        if number % i == 0:
            sum_of_divisors += i

    # Check if the sum of divisors equals the original number
    return sum_of_divisors == number
# Example Usage:
print(is_perfect_number(6))    # True
print(is_perfect_number(28))   # True
print(is_perfect_number(12))   # False
print(is_perfect_number(496))  # True
```

```
True
True
False
True
```

Justification :

This task is significant because it examines how well the AI can solve the perfect number problem using only a zero-shot prompt. The generated code usually implements the core divisor-sum logic correctly, but may include inefficiencies such as unnecessary full-range loops or missing checks for non-positive inputs. This shows that zero-shot prompting is effective for generating basic solutions but not for optimized or production-level code. It emphasizes the need for refined prompts when performance and correctness are important.

Task 6: Few-Shot Prompting (Even or Odd Classification with Validation)

PROMPT :

Generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd

• Input: 0 → Output: Even

CODE and OUTPUT :

```python
def check_even_odd(number):
    """
    Determine whether a given number is even or odd.

    Parameters:
    number (int): The number to check.

    Returns:
    str: "Even" if the number is even, "Odd" if the number is odd.
    """
    # Input validation
    if not isinstance(number, int):
        return "Input must be an integer."

    # Check if the number is even or odd
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Example Usage:
print(check_even_odd(8))    # Output: Even
print(check_even_odd(15))   # Output: Odd
print(check_even_odd(0))    # Output: Even
print(check_even_odd(-4))   # Output: Even
```

```
Even
Odd
Even
Even
```

Justification :

Few-shot prompting is significant because multiple examples train the AI to recognize patterns and apply correct validation logic. Including cases like zero guides the AI to produce consistent and meaningful outputs. The generated code is more structured and user-friendly, with better conditional handling. Compared to zero-shot prompting, fewer logical errors are observed. This confirms that few-shot prompting produces more robust and dependable programs.