Assignment-3.1

NAME: E.Pavani

ROLL NO: 2303A52445

COURSE: AI Assisted Coding

Question 1: Zero-Shot Prompting (Palindrome Number Program) Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.
Task:

• Record the AI-generated code.

• Test the code with multiple inputs.

• Identify any logical errors or missing edge-case handling.

PROMPT

```python
# # Write a Python function that checks whether a given number is a palindrome.
# The function should:
# - Accept a single integer as input
# - Return True if the number is a palindrome, False otherwise
# - A palindrome number reads the same forwards and backwards
# - Handle negative numbers appropriately
# - The function name should be "is_palindrome"
# - Include a docstring explaining what the function does
# - Provide the complete, working function code
def is_palindrome(num):
    """Check if a number is a palindrome."""
    str_num = str(num)
    return str_num == str_num[::-1]
num=int(input("Enter a number to check if it is a palindrome: "))
if is_palindrome(num):
    print(f"{num} is a palindrome.")
else:
    print(f"{num} is not a palindrome.")
```

OUTPUT

```
ive\Desktop\AI ASSISSTENT\ass-3.1.py'
Enter a number to check if it is a palindrome: 121
121 is a palindrome.
```

ANALYSISI

- The zero-shot prompt gave a correct program for checking palindrome numbers.
- The program works well for normal positive numbers.
- It does not clearly handle negative numbers or wrong input types.
- This shows that the code works, but some special cases are missing.

Question 2: One-Shot Prompting (Factorial Calculation) 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.

Example:

Input: 5 → Output: 120

Task:

• Compare the generated code with a zero-shot solution.

• Examine improvements in clarity and correctness.

PROMPT

```python
# Write a Python function to compute the factorial of a given number.

# Requirements:
# - Function name: "factorial"
# - Accept a single integer parameter
# - Return the factorial value as an integer
# - Handle edge cases appropriately
# - Include comprehensive docstring
# - Provide complete, working code

def factorial(n):
    """
    Compute the factorial of a given number.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0 or n == 1:
        return 1

    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
n=int(input("Enter a number to compute its factorial: "))
if n>=0:
    print(f"The factorial of {n} is {factorial(n)}.")
else:
    print("Factorial is not defined for negative numbers.")
```

OUTPUT

```
\OneDrive\Desktop\AI ASSISSTENT\ass-3.1.py'
Enter a number to compute its factorial: 5
The factorial of 5 is 120.
```

ANALYSIS

- One-shot prompting produces better and safer code than zero-shot prompting.
- The example helps the AI write clearer logic and handle edge cases correctly.

Question 3: Few-Shot Prompting (Armstrong Number Check) 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.

Examples:

• Input: 153 → Output: Armstrong Number

• Input: 370 → Output: Armstrong Number

• Input: 123 → Output: Not an Armstrong Number

Task:

• Analyze how multiple examples influence code structure and

accuracy.

• Test the function with boundary values and invalid inputs.

PROMPT

```
# Write a Python function to check if a number is an Armstrong Number

# DEFINITION:
# An Armstrong number (also called narcissistic number) is a number that equals
# the sum of its own digits each raised to the power of the number of digits.

# MULTIPLE EXAMPLES (Few-Shot):

# Example 1 (Armstrong - 3 digits):
# Input: 153
# Output: "Armstrong Number"
# Verification: 1³ + 5³ + 3³ = 1 + 125 + 27 = 153 ✓

# Example 2 (Armstrong - 3 digits):
# Input: 370
# Output: "Armstrong Number"
# Verification: 3³ + 7³ + 0³ = 27 + 343 + 0 = 370 ✓

# Example 3 (Not Armstrong - 3 digits):
# Input: 123
# Output: "Not an Armstrong Number"
# Verification: 1³ + 2³ + 3³ = 1 + 8 + 27 = 36 ≠ 123 ✗

# REQUIREMENTS:
# - Function name: "check_armstrong"
# - Accept a single integer parameter
# - Return "Armstrong Number" for Armstrong numbers
# - Return "Not an Armstrong Number" for non-Armstrong numbers
# - Handle edge cases: 0, 1, single digits, negative numbers
# - Include docstring with examples
# - Provide complete, working code
```

```python
def check_armstrong(num):

    if num < 0:
        return "Not an Armstrong Number"

    digits = str(num)
    num_digits = len(digits)
    sum_of_powers = sum(int(digit) ** num_digits for digit in digits)

    if sum_of_powers == num:
        return "Armstrong Number"
    else:
        return "Not an Armstrong Number"
n=int(input("Enter a number to check if it is an Armstrong number: "))
result = check_armstrong(n)
print(f"{n} is {result}.")
```

OUTPUT

```
Enter a number to check if it is an Armstrong number: 886
886 is Not an Armstrong Number.
```

ANALYSIS

- The code structure is better and follows the examples.
- The code does not handle negative or invalid inputs properly.

- Accuracy is higher compared to zero-shot and one-shot prompting.

Question 4: Context-Managed Prompting (Optimized Number Classification) 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.

Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.

PROMPT

```
# CONTEXT & CONSTRAINTS:
# 1. INPUT VALIDATION:
#    • Must accept only integers (reject floats, strings, None)
#    • Range: -2147483648 to 2147483647 (32-bit signed integer)
#    • Raise ValueError for invalid inputs with descriptive message
#    • Return None for out-of-range values

# 2. CLASSIFICATION RULES:
#    • Prime: Integer > 1 with no divisors except 1 and itself
#    • Composite: Integer > 1 that is NOT prime
#    • Neither: Integers ≤ 1 (0, 1, and all negative numbers)

# 3. OPTIMIZATION REQUIREMENTS:
#    • Use mathematical shortcuts (e.g., check divisibility only up to √n)
#    • Avoid unnecessary computations
#    • Skip even numbers after checking 2
#    • Time complexity should be O(√n) for prime checking
#    • Space complexity should be O(1)

# 4. OUTPUT FORMAT & CONSISTENCY:
#    • Return tuple: (number, classification, is_valid)
#    • classification: "Prime", "Composite", or "Neither"
#    • is_valid: True if input is valid integer, False otherwise
#    • Raise ValueError ONLY for non-integer inputs, not for out-of-range

# 5. PERFORMANCE CONSIDERATIONS:
#    • Cache small primes for quick lookup (2, 3, 5, 7)
#    • Use bitwise operations where applicable
#    • Minimize function calls in loops
#    • Profile for numbers: 1, 2, 3, 10, 100, 1000, 1000000

# 6. EDGE CASES TO HANDLE:
#    • Negative numbers: Classify as "Neither"
#    • Zero: Classify as "Neither"
#    • One: Classify as "Neither"
#    • Two: Classify as "Prime"
#    • Large primes: e.g., 1000000007, 999983
#    • Powers of 2: Must be recognized as composite (except 2 itself)

# EXAMPLES:
# ─────────
# Input: 17 → Output: (17, "Prime", True)        [17 is prime]
# Input: 24 → Output: (24, "Composite", True)    [24 is divisible by 2]
# Input: 1  → Output: (1, "Neither", True)       [1 is special case]
# Input: -5 → Output: (-5, "Neither", True)      [Negative is neither]
# Input: 3.5 → Raise ValueError: "Input must be integer"
# Input: "text" → Raise ValueError: "Input must be integer"
```

```python
def is_prime_optimized(n: int) -> bool:
    """Check if a number is prime using optimized method."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
def classify_number_optimized(num: int) -> tuple[int, str, bool]:
    if not isinstance(num, int):
        raise ValueError("Input must be an integer")

    if num < -2147483648 or num > 2147483647:
        return (num, "Neither", False)

    if num <= 1:
        return (num, "Neither", True)
    elif is_prime_optimized(num):
        return (num, "Prime", True)
    else:
        return (num, "Composite", True)
n=int(input("Enter a number to classify: "))
if isinstance(n, int):
    result = classify_number_optimized(n)
    print(f"Classification of {n}: {result}")
else:
    print("Input must be an integer.")
```

OUTPUT

```
Enter a number to classify: 143
Classification of 143: (143, 'Composite', True)
================================================================
```

ANALYSIS

- Input validation is properly handled.
- Compared to earlier prompting methods, this gives the best accuracy and performance**.**
- Clear instructions helped the AI write correct and efficient code.

Question 5: Zero-Shot Prompting (Perfect Number Check) Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

• Record the AI-generated code.

• Test the program with multiple inputs.

• Identify any missing conditions or inefficiencies in the logic.

PROMPT

```python
# Write a Python function that checks whether 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 the number itself).

# REQUIREMENTS:
# - Function name: "is_perfect_number"
# - Accept a single integer parameter
# - Return True if the number is perfect, False otherwise
# - Handle edge cases appropriately
# - Include a docstring
# - Provide complete, working code
def is_perfect_number(num):
    """Check if a number is a perfect number."""
    if num <= 0:
        return False

    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum == num
n=int(input("Enter a number to check if it is a perfect number: "))
if is_perfect_number(n):
    print(f"{n} is a perfect number.")
else:
    print(f"{n} is not a perfect number.")
```

OUTPUT

```
Enter a number to check if it is a perfect number: 778
778 is not a perfect number.
```

ANALYSIS

The zero-shot prompt created a correct program for checking perfect numbers.
The program works well for normal positive numbers like 6 and 28.
It does not handle 0, negative numbers, or wrong inputs properly.

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

Examples:

• Input: 8 → Output: Even

• Input: 15 → Output: Odd

• Input: 0 → Output: Even

Task:

• Analyze how examples improve input handling and output

clarity.

• Test the program with negative numbers and non-integer inputs.

PROMPT

```
# Write a Python function that determines whether a given number is even or odd,
# including proper input validation.

# MULTIPLE EXAMPLES:

# Example 1 (Even - Positive):
# Input: 8
# Output: "Even"
# Explanation: 8 is divisible by 2 with no remainder

# Example 2 (Odd - Positive):
# Input: 15
# Output: "Odd"
# Explanation: 15 divided by 2 leaves remainder of 1

# Example 3 (Zero - Special Even Case):
# Input: 0
# Output: "Even"
# Explanation: 0 is considered even (0 % 2 == 0)

# REQUIREMENTS:
# - Function name: "classify_even_odd"
# - Accept a single parameter (should be integer)
# - Return "Even" or "Odd" as strings
# - Handle input validation (reject non-integers)
# - Raise ValueError with descriptive message for invalid inputs
# - Handle edge cases: negative numbers, zero
# - Include comprehensive docstring with examples
# - Provide complete, working code
```

```python
def classify_even_odd(num):
    """
    Classify a number as 'Even' or 'Odd'.

    Examples:
    >>> classify_even_odd(8)
    'Even'
    >>> classify_even_odd(15)
    'Odd'
    >>> classify_even_odd(0)
    'Even'

    Args:
        num (int): The number to classify.

    Returns:
        str: "Even" if the number is even, "Odd" if odd.

    Raises:
        ValueError: If input is not an integer.
    """
    if not isinstance(num, int):
        raise ValueError("Input must be an integer")

    return "Even" if num % 2 == 0 else "Odd"
n=int(input("Enter a number to classify as even or odd: "))
if isinstance(n, int):
    result = classify_even_odd(n)
    print(f"{n} is {result}.")
else:
    print("Input must be an integer.")
```

OUTPUT

```
Enter a number to classify as even or odd: 56
56 is Even.
```

ANALYSIS

The given examples helped the AI understand what even and odd mean.
The program gives clear output like "Even" or "Odd".
It correctly works for negative numbers also.
It safely rejects non-integer inputs like decimals or strings.