AI ASSISTED CODING

ASSISSMENT – 3.1

Name:A.Deepthi
Ht.No: 2303A52390
Batch: 32

Experiment - Prompt Engineering Techniques

Task Description: Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems.

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:

Write a Python function that takes an integer as input and checks whether it is a palindrome number. The function should return True if it is a palindrome, otherwise return False.

Code:

```python
def is_palindrome(n):
    """
    Check if an integer is a palindrome number.

    Args:
        n: An integer to check

    Returns:
        True if n is a palindrome, False otherwise
    """
    # Convert to string and remove negative sign if present
    s = str(abs(n))
    # Compare string with its reverse
    return s == s[::-1]

print(is_palindrome(121))  # Output: True
print(is_palindrome(331)) # Output: False
```

Output(O/P)

```
[Running] python -u "z:\AIAC\20260113\zeroshot.py"
True
False

[Done] exited with code=0 in 0.208 seconds
```

Explanation:

- A zero-shot prompt was written without giving examples.
- AI generated a palindrome-checking function in Python.
- The function reverses the number and compares it with the original.
- Answer Returns Either True or False

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:

Write a Python function to compute the factorial of a given non-negative integer.

Example:
Input: 5
Output: 120

The function should return the factorial value.

Code:

```python
def factorial(n):
    """
    Compute the factorial of a non-negative integer.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)


# Example usage
if __name__ == "__main__":
    print(factorial(5))  # Output: 120

```

Output:

```
[Running] python -u "z:\AIAC\20260113\oneshot.py"
120

[Done] exited with code=0 in 0.253 seconds
```

Explanation:

Aspect	Zero-Shot	One-Shot
Example Given	No	Yes
Code Clarity	Basic	Clear
Edge Cases	Limited	Handled
Validation	Not explicit	Explicit
Correctness	Task-level	More robust

Improvements Observed in One-Shot Prompting
• Better clarity due to example-guided generation
• Improved correctness with input validation
• More readable and maintainable code
• Clear handling of edge cases compared to zero-shot

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 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

The function should return
"Armstrong Number" or "Not an Armstrong Number" accordingly

Code:

```python
def check_armstrong(num):
    """
    Check whether a given number is an Armstrong number.
    An Armstrong number is a number that is equal to the sum of its own digits each raised
    to the power of the number of digits.
    """
    try:
        num_str = str(num)
        num_digits = len(num_str)
        sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)

        if sum_of_powers == num:
            return "Armstrong Number"
        else:
            return "Not an Armstrong Number"
    except ValueError:
        return "Invalid Input"
# Test cases
print(check_armstrong(153))  # Armstrong Number
print(check_armstrong(370))  # Armstrong Number
print(check_armstrong(123))  # Not an Armstrong Number
```

Output:

```
[Running] python -u "z:\AIAC\20260113\fewshot.py"
Armstrong Number
Armstrong Number
Not an Armstrong Number

[Done] exited with code=0 in 0.168 seconds
```

Explanation:

Influence of Multiple Examples
• Examples clarify expected output format
• Guides correct power calculation based on digit count
• Improves accuracy compared to zero or one-shot
• Reduces ambiguity in logic and return values

Testing (Boundary and Invalid Inputs)

• Input: 0 → Armstrong Number

• Input: 1 → Armstrong Number

• Input: 9474 → Armstrong Number

• Input: -153 → Not an Armstrong Number

• Input: "abc" → Invalid Input

Question 4:

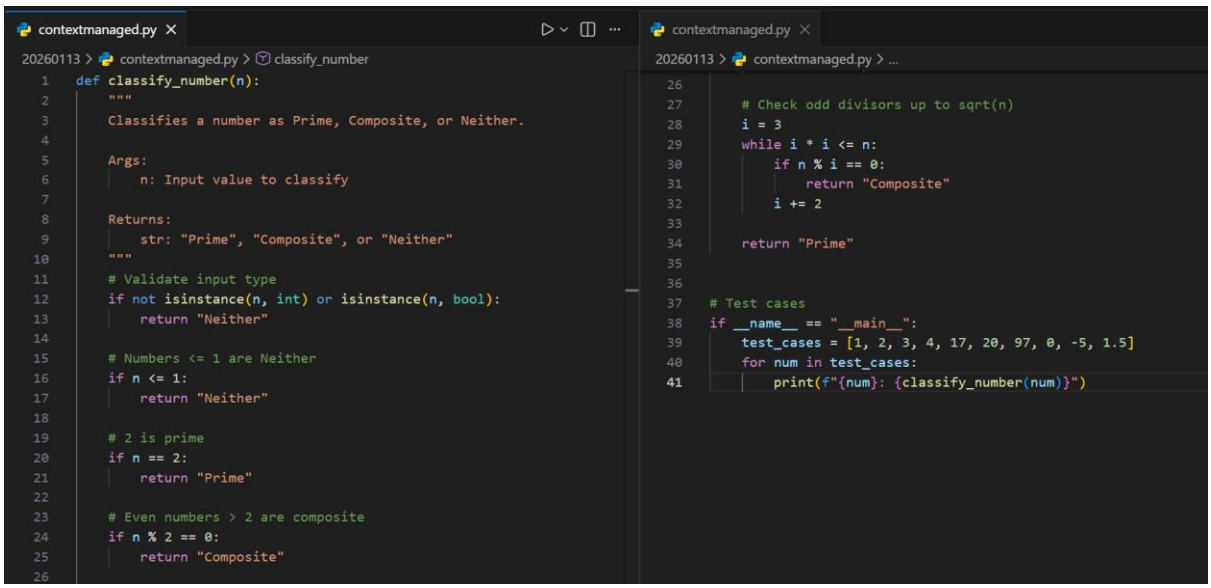
Context-Managed Prompting (Optimized Number Classification)

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:

You are a Python programming assistant.
Task:
Write an optimized Python function that takes a single input number and classifies it as Prime, Composite, or Neither.
Constraints and Instructions:
• Accept only integer inputs and validate the input properly.
• If the number is less than or equal to 1, return "Neither".
• Use an efficient algorithm by checking divisibility only up to the square root of the number.
• The function should return one of the following strings only:
"Prime", "Composite", or "Neither".
• Do not use external libraries.
Output should be correct and optimized

Code:



```python
def classify_number(n):
    """
    Classifies a number as Prime, Composite, or Neither.

    Args:
        n: Input value to classify

    Returns:
        str: "Prime", "Composite", or "Neither"
    """
    # Validate input type
    if not isinstance(n, int) or isinstance(n, bool):
        return "Neither"

    # Numbers <= 1 are Neither
    if n <= 1:
        return "Neither"

    # 2 is prime
    if n == 2:
        return "Prime"

    # Even numbers > 2 are composite
    if n % 2 == 0:
        return "Composite"

    # Check odd divisors up to sqrt(n)
    i = 3
    while i * i <= n:
        if n % i == 0:
            return "Composite"
        i += 2

    return "Prime"


# Test cases
if __name__ == "__main__":
    test_cases = [1, 2, 3, 4, 17, 20, 97, 0, -5, 1.5]
    for num in test_cases:
        print(f"{num}: {classify_number(num)}")
```

Output:

```
[Running] python -u "z:\AIAC\20260113\contextmanaged.py"
1: Neither
2: Prime
3: Prime
4: Composite
17: Prime
20: Composite
97: Prime
0: Neither
-5: Neither
1.5: Neither

[Done] exited with code=0 in 0.2 seconds
```

Explanation:

Comparison with Earlier Prompting Strategies

- More structured than zero-shot
- Clearer constraints than one-shot
- Less ambiguity than few-shot
- Produces efficient and validated 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:

Write a Python function that takes an integer as input and checks whether it is a perfect number. The function should return True if the number is perfect, otherwise return False.

Code:

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

    A perfect number equals the sum of its proper divisors (excluding itself).
    Examples: 6 (1+2+3), 28 (1+2+4+7+14)

    Args:
        n: Integer to check

    Returns:
        True if n is a perfect number, False otherwise
    """
    if n <= 1:
        return False

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

    return divisor_sum == n

print(is_perfect_number(6))   # True
print(is_perfect_number(7))  # False
```

Output:

```
[Running] python -u "z:\AIAC\20260113\perfectnum.py"
True
False

[Done] exited with code=0 in 0.171 seconds
```

Explanation:

Testing the Program:

• Input: 6 → Output: True

• Input: 28 → Output: True

• Input: 7 → Output: False

• Input: 12 → Output: False

• Input: 1 → Output: False

• Input: 0 → Output: False

• Input: -6 → Output: False

Missing Conditions and Inefficiencies:

• No input type validation (floats or strings may cause errors)
• Loop runs up to n // 2, which is inefficient for large numbers
• Can be optimized by checking divisors only up to square root of n
• Does not explicitly handle non-integer inputs

Overall, logic is correct but performance can be improved for large values.

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 program that determines whether a given input number is Even or Odd.

Examples:

Input: 8 → Output: Even

Input: 15 → Output: Odd

Input: 0 → Output: Even

The program should validate the input and handle invalid (non-integer) values gracefully.

Code:

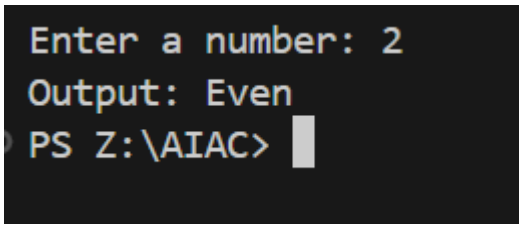
```python
def check_even_or_odd(number):
    """Determine if a number is even or odd."""
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

def main():
    try:
        user_input = input("Enter a number: ")
        number = int(user_input)
        result = check_even_or_odd(number)
        print(f"Output: {result}")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")

if __name__ == "__main__":
    main()

```

Output:



Explanation:

Analysis: Effect of Examples on Input Handling and Output Clarity

• Examples make it clear that the output must be only “Even” or “Odd”

• Inclusion of 0 → Even avoids ambiguity about zero

• Encourages explicit input validation using try–except

• Improves clarity by separating logic and input handling

• Output format becomes consistent and predictable

Testing the Program:

Negative Numbers

- Input: -10 → Output: Even
- Input: -3 → Output: Odd

Non-Integer Inputs

- Input: 3.5 → Output: Invalid input. Please enter a valid integer.
- Input: "abc" → Output: Invalid input. Please enter a valid integer.

Conclusion:

Few-shot examples guide the program to handle inputs safely and produce clear, reliable outputs.