AI ASSISTED CODING

LAB ASSIGNMENT – 3.1

Name : Sarayu

Hall Ticket Number : 2303a51842

Question 1: Zero-Shot Prompting – Palindrome Number Program

Prompt Used

Write a Python function to check whether a given number is a palindrome.

AI-Generated Code

```
def is_palindrome(num):
    temp = num
    rev = 0
    while num > 0:
        digit = num % 10
        rev = rev * 10 + digit
        num //= 10
    return temp == rev
```

Testing (Input → Output)

Input: 121 → Output: True Input: 123 → Output: False Input: 0 → Output: True Input: -121 → Output: False.

Conclusion

Zero-shot prompting provides a basic working solution but lacks comprehensive edge-case handling.

Question 2: One-Shot Prompting – Factorial Calculation

Prompt Used

Write a Python function to calculate the factorial of a number. Example: Input 5 → Output 120.

AI-Generated Code

```
def factorial(n):
    if n < 0:
        return "Invalid input"
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Testing (Input → Output)

Input: 5 → Output: 120 Input: 0 → Output: 1 Input: 1 → Output: 1 Input: -3 → Output: Invalid input

Observations

The program correctly handles negative inputs and computes factorial for zero and positive values.

Conclusion
One-shot prompting improves correctness and clarity compared to zero-shot prompting.

Question 3: Few-Shot Prompting – Armstrong Number Check

Prompt Used
Write a Python function to check whether a number is an Armstrong number using examples.

AI-Generated Code

```
def is_armstrong(num):
    if num < 0:
        return "Invalid input"
    digits = str(num)
    power = len(digits)
    total = 0
    for d in digits:
        total += int(d) ** power
    return "Armstrong Number" if total == num else "Not an Armstrong Number"
```

Testing (Input → Output)
Input: 153 → Output: Armstrong Number Input: 370 → Output: Armstrong Number Input: 123 → Output: Not an Armstrong Number Input: 0 → Output: Armstrong Number Input: -10 → Output: Invalid input

Observations
The logic is accurate and handles boundary cases such as zero correctly.

Conclusion
Few-shot prompting significantly enhances accuracy and logical structure.

Question 4: Context-Managed Prompting – Optimized Number Classification

Prompt Used
Write an optimized Python program to classify a number as Prime, Composite, or Neither..

AI-Generated Code

```
def classify_number(n):
    if not isinstance(n, int):
        return "Invalid input"
    if n <= 1:
        return "Neither Prime nor Composite"
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return "Composite"
    return "Prime"
```

Testing (Input → Output)

Input: 2 → Output: Prime Input: 9 → Output: Composite Input: 1 → Output: Neither Prime nor Composite Input: -5 → Output: Neither Prime nor Composite Input: 7 → Output: Prime

Observations

The program is efficient and correctly validates invalid inputs.

Conclusion

Context-managed prompting produces optimized and reliable code.

Question 5: Zero-Shot Prompting – Perfect Number Check

Prompt Used

Write a Python function to check whether a number is a perfect number..

AI-Generated Code

```
def is_perfect(num):
    if num <= 0:
        return False
    total = 0
    for i in range(1, num):
        if num % i == 0:
            total += i
    return total == num
```

Testing (Input → Output)

Input: 6 → Output: True Input: 28 → Output: True Input: 12 → Output: False Input: 1 → Output: False

Observations

The logic is correct but inefficient for large numbers due to linear iteration.

Conclusion

Zero-shot prompting lacks performance optimization.

Question 6: Few-Shot Prompting – Even or Odd Classification with Validation

Prompt Used

Write a Python program to determine whether a number is even or odd using examples..

AI-Generated Code

```
def even_or_odd(n):
    if not isinstance(n, int):
        return "Invalid input"
    return "Even" if n % 2 == 0 else "Odd"
```

Testing (Input → Output)

Input: 8 → Output: Even Input: 15 → Output: Odd Input: 0 → Output: Even Input: -4 → Output: Even Input: 3.5 → Output: Invalid input

Conclusion
Few-shot prompting improves input handling and output consistency.

Overall Conclusion

This experiment clearly demonstrates that prompt engineering techniques play a vital role in determining the quality of AI-generated code. Zero-shot prompting provides basic solutions, one-shot prompting improves clarity, few-shot prompting enhances accuracy, and context-managed prompting delivers optimized and production-level solutions.