

Assignment - 3.1

Hall Ticket - 2303A51178

Batch - 29

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

Prompt Used (Zero-Shot) - Write a Python function that checks whether a given number is a palindrome, and analysis at the end.

```
def is_palindrome(num):
    original = num
    reverse = 0

    while num > 0:
        digit = num % 10
        reverse = reverse * 10 + digit
        num //= 10

    return original == reverse
```

AI-Generated Code -

```
print(is_palindrome(121))    # True
print(is_palindrome(123))    # False
print(is_palindrome(1221))   # True
```

```
True
False
True
```

Output -

Analysis -

- The code correctly identifies palindrome numbers.
- Negative numbers are not handled properly.
- Zero is handled correctly.
- No input validation is present.
- Zero-shot prompting produced correct logic but missed edge-case handling.

Question 2: One-Shot Prompting (Factorial Calculation)

Prompt Used (One-Shot) - Write a Python function to calculate the factorial of a number.

Example -

Input: 5

Output: 120

AI-Generated

```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Code -

Test Cases & Output -

```
print(factorial(5))
```

120

Comparison with Zero-Shot -

- One-shot prompting improved clarity.
- Input validation was included.
- Logic is clearer and more reliable.
- Example helped AI infer edge cases.

Question 3: Few-Shot Prompting – Armstrong Number

Prompt Used (Few-Shot) - Write a Python function to check whether a number is an Armstrong number.

Examples:

Input: 153 → Output: Armstrong Number

Input: 370 → Output: Armstrong Number

Input: 123 → Output: Not an Armstrong Number

AI-

Code -

```
def is_armstrong(num):  
    temp = num  
    digits = len(str(num))  
    total = 0  
  
    while temp > 0:  
        digit = temp % 10  
        total += digit ** digits  
        temp //= 10  
  
    if total == num:  
        return "Armstrong Number"  
    else:  
        return "Not an Armstrong Number"
```

Generated

Test Cases &

```
print(is_armstrong(153)) # Armstrong Number  
print(is_armstrong(370)) # Armstrong Number  
print(is_armstrong(123)) # Not an Armstrong Number
```

Output -

```
Armstrong Number  
Armstrong Number  
Not an Armstrong Number
```

Analysis -

- Few-shot examples significantly improved accuracy.
- Code structure is clearer.
- Negative and non-integer inputs are not validated.
- Multiple examples helped AI understand the pattern correctly.

Question 4: Context-Managed Prompting – Prime / Composite / Neither (Optional)

Prompt Used (Context-Managed) - Write an optimized Python program to classify a number as prime, composite, or neither.

Ensure proper input validation and efficient logic.

AI-

```
def classify_number(n):
    # Input validation
    if not isinstance(n, int):
        return "Invalid input: Please enter an integer"

    if n <= 1:
        return "Neither Prime nor Composite"

    # Efficient prime check (up to √n)
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return "Composite Number"

    return "Prime Number"
```

Generated Code -

Test

```
print(classify_number(1))    # Neither Prime nor Composite
```

Cases &

```
print(classify_number(2))    # Prime Number
```

Output -

```
print(classify_number(15))   # Composite Number
```

```
print(classify_number(17))   # Prime Number
```

Neither Prime nor Composite

Prime Number

Composite Number

Prime Number

Analysis -

- Context-managed prompt resulted in optimized logic.
- Efficient square-root approach used.
- Proper handling of special cases.
- Best result among all prompting strategies.

Question 5: Zero-Shot Prompting – Perfect Number

Prompt Used (Zero-Shot) - Write a Python function to check whether a number is a perfect number.

AI-

Code -

Generated

```
def is_perfect_number(n):
    # Input validation
    if not isinstance(n, int) or n <= 0:
        return "Invalid input"

    if n == 1:
        return "Not a Perfect Number"

    divisor_sum = 1 # 1 is always a proper divisor

    # Check divisors efficiently up to √n
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            divisor_sum += i
            if i != n // i:
                divisor_sum += n // i

    if divisor_sum == n:
        return "Perfect Number"
    else:
        return "Not a Perfect Number"
```

Test Cases**&****Output -**

```
print(is_perfect_number(6))      # Perfect Number
print(is_perfect_number(28))      # Perfect Number
print(is_perfect_number(12))      # Not a Perfect Number
```

Perfect Number
Perfect Number
Not a Perfect Number

Analysis -

- Logic is correct.
- Inefficient for large numbers.
- No optimization applied.
- Zero-shot prompt lacked performance considerations.

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

Prompt Used (Few-Shot) - Write a Python program to check whether a number is even or odd.

Examples:

Input: 8 → Even

Input: 15 → Odd

Input: 0 → Even

AI-Generated

```
def check_even_odd(n):
    # Input validation
    if not isinstance(n, int):
        return "Invalid input"

    if n % 2 == 0:
        return "Even"
    else:
        return "Odd"
```

Code -

Test Cases &

```
print(check_even_odd(8))    # Even
print(check_even_odd(15))    # Odd
print(check_even_odd(0))     # Even
```

Output -

Even
Odd
Even

Analysis -

- Few-shot examples improved input handling.
- Clear output formatting.
- Proper validation included.
- Best suited for real-world usage.