

# ASSIGNMENT-3.1

**Name:** Nandini Reddy

**HT. No:** 2303A51896

**Batch:** 08

Lab Experiment: Prompt Engineering – Improving Prompts and Context Management.

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.

```
1 #Question 1:generate a python program that checks whether a given number is a
  palindrome
2 def is_palindrome(num):
3     str_num = str(num)
4     return str_num == str_num[::-1]
5 print(is_palindrome(121))
6 print(is_palindrome(123))
```

```
True
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.

Task:

- Compare the generated code with a zero-shot solution.
- Examine improvements in clarity and correctness.

```
1 # #QUESTION 2 : write a python program that compute factorial of a number
2 def factorial(n):
3     if n == 0 or n == 1:
4         return 1
5     else:
6         return n * factorial(n - 1)
7 print(factorial(5))
```

```
120
```

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.

Task:

- Analyze how multiple examples influence code structure and

accuracy.

- Test the function with boundary values and invalid inputs. (Optional Extension)

```
1 # #QUESTION 3 : write a python program that checks given number is an armstrong or not
2 # #output should be like Armstrong number or Not an Armstrong number
3 def is_armstrong(num):
4     str_num = str(num)
5     num_digits = len(str_num)
6     sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
7     if sum_of_powers == num:
8         return "Armstrong number"
9     else:
10        return "Not an Armstrong number"
11 print(is_armstrong(153))
12 print(is_armstrong(370))
13 print(is_armstrong(123))
```

```
Armstrong number
Armstrong number
Not an Armstrong number
```

#### 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.

```
1 #QUESTION 4 : to generate an optimized Python program that classifies a number as prime, composite, or neither
2 def classify_number(num):
3     if num <= 1:
4         return "Neither prime nor composite"
5     for i in range(2, int(num**0.5) + 1):
6         if num % i == 0:
7             return "Composite"
8     return "Prime"
9 print(classify_number(11))
10 print(classify_number(15))
11 print(classify_number(1))
```

```
Prime
Composite
Neither prime nor composite
```

#### 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.

```
1 #QUESTION 5 : generate a Python function that checks whether a given number is
2 # a perfect number
3 def is_perfect_number(num):
4     if num < 1:
5         return False
6     divisors_sum = sum(i for i in range(1, num) if num % i == 0)
7     return divisors_sum == num
8 print(is_perfect_number(6))
9 print(is_perfect_number(28))
10 print(is_perfect_number(12))
11 print(is_perfect_number(14))
```

```
True
True
False
False
```

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.

Task:

- Analyze how examples improve input handling and output clarity.
- Test the program with negative numbers and non-integer inputs.

```
1 #Question 6 : generate a Python program that determines whether a given number
2 # is even or odd, including proper input validation
3 #output should be like Even or Odd
4 def is_even_or_odd(num):
5     if not isinstance(num, int):
6         return "Invalid input. Please enter an integer."
7     return "Even" if num % 2 == 0 else "Odd"
8 print(is_even_or_odd(8))
9 print(is_even_or_odd(15))
10 print(is_even_or_odd(0))
```

```
Even
Odd
Even
```