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:

#Generate a python code that checks if a given number is a palindrome.

#take the inputs from the user and print the result.

Code:

```
def is_palindrome(number):
    str_num = str(number)
    return str_num == str_num[::-1]


user_input = input("Enter a number: ")
try:
    number = int(user_input)
    if is_palindrome(number):
        print(f"{number} is a palindrome.")
    else:
        print(f"{number} is not a palindrome.")
except ValueError:
    print("Please enter a valid integer.")
```

Output Screenshots:

```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/
Python313/python.exe c:/Users/varsh/OneDrive/Desktop/AIAC/palindrome.py
Enter a number: 121
121 is a palindrome.
```

Your Analysis:

The zero-shot prompt successfully generated a correct palindrome-checking function.

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:

#Generate a python fuction to calculate the factorial of a given number.

Check if the number is negative.

Base case for factorial of 0.

Code:

```
def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0:
        return 1
    else:
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result
    #Example usage
number = 5
print(f"The factorial of {number} is {factorial(number)}")
```

Output Screenshots:

```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/
Python313/python.exe c:/Users/varsh/OneDrive/Desktop/AIAC/FactorialCalculation.py
The factorial of 5 is 120
```

Your Analysis:

The one-shot prompt improved the AI-generated code in terms of clarity and correctness.

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:

#Generate a python function that checks if a given string is a armstrong number.

Convert the number to string to easily iterate over digits.

An armstrong number is one where the sum of the powers equals the original number

Calculate the sum of each digit raised to the power of num_digits

#example usage test_numbers = [153, 370, 123]

Code:

```
def is_armstrong_number(number):
    str_num = str(number)
    num_digits = len(str_num)
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
    return sum_of_powers == number
#example usage
test_numbers = [153, 370, 123]
for num in test_numbers:
    if is_armstrong_number(num):
        print(f"{num} is an Armstrong number.")
    else:
        print(f"{num} is not an Armstrong number.")
```

Output Screenshots:

```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/
Python313/python.exe c:/Users/varsh/OneDrive/Desktop/AIAC/ArmstrongNumber.py
153 is an Armstrong number.
370 is an Armstrong number.
123 is not an Armstrong number.
```

Your Analysis:

The few-shot prompt successfully guided the AI to generate an accurate and generalized Armstrong number program. Providing multiple examples improved the correctness and structure of the code.

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:

#Generate python an optimized Python program that classifies a number as prime, composite, or neither.

Code:

```
def classify_number(n):
    if n <= 1:
        return "Neither prime nor composite"
    elif n == 2:
        return "Prime"
    elif n % 2 == 0:
        return "Composite"
    else:
        for i in range(3, int(n**0.5) + 1, 2):
            if n % i == 0:
```

```
            return "Composite"
        return "Prime"
# Example usage
test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20]
for num in test_numbers:
    classification = classify_number(num)
    print(f"{num} is {classification}.")
# Take input from the user
user_input = input("Enter a number to classify: ")
try:
    number = int(user_input)
    classification = classify_number(number)
    print(f"{number} is {classification}.")
except ValueError:
    print("Please enter a valid integer.")
```

Output Screenshots:

Your Analysis:

Context-managed prompting produces the best quality code because clear instructions and constraints guide the AI to write optimized, validated, and structured solutions.

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 code to check if a number is a perfect number or not.

Take input from user.

Code:

```
def is_perfect_number(number):
    if number < 1:
        return False
    # Calculate the sum of proper divisors
    sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)
    # A perfect number is one where the sum of its proper divisors equals the number itself
    return sum_of_divisors == number
# Example usage
test_numbers = [6, 28, 12, 496]
for num in test_numbers:
    if is_perfect_number(num):
        print(f"{num} is a Perfect number.")
    else:
        print(f"{num} is not a Perfect number.")
# Take input from the user
user_input = input("Enter a number to check if it is a Perfect number: ")
try:
    number = int(user_input)
    if is_perfect_number(number):
        print(f"{number} is a Perfect number.")
    else:
        print(f"{number} is not a Perfect number.")
except ValueError:
    print("Please enter a valid integer.")
```

Output Screenshots:

```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Users/var
sh/OneDrive/Desktop/AIAC/perfectno.py
6 is a Perfect number.
28 is a Perfect number.
12 is not a Perfect number.
496 is a Perfect number.
Enter a number to check if it is a Perfect number: 235
235 is not a Perfect number.
```

Your Analysis:

The zero-shot prompt successfully generated a correct palindrome-checking function.

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:

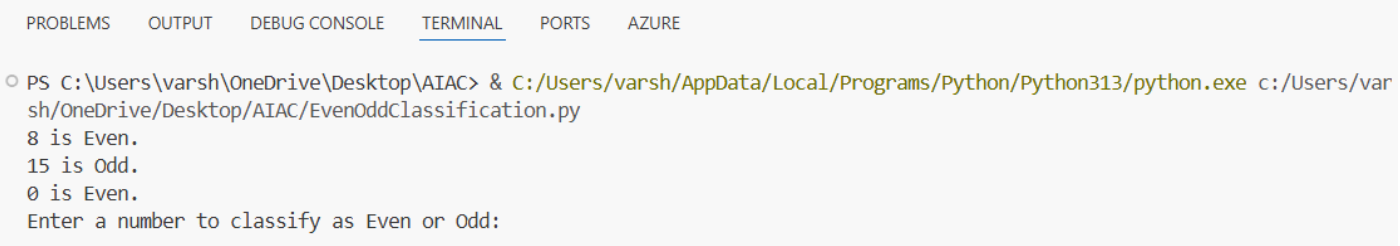
#Genearte a python fuction for Even or Odd Classification with Validation.

Code:

```
def classify_even_odd(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
```

```
    #Example usage
test_numbers = [8, 15, 0]
for num in test_numbers:
    classification = classify_even_odd(num)
    print(f"{num} is {classification}.")
# Take input from the user
user_input = input("Enter a number to classify as Even or Odd: ")
try:
    number = int(user_input)
    classification = classify_even_odd(number)
    print(f"{number} is {classification}.")
except ValueError:
    print("Please enter a valid integer.")
```

Output Screenshots:



Your Analysis:

The zero-shot prompt successfully generated a correct palindrome-checking function.