Assginment-3.1

Name :P.Akshith

Batch No : 47-A

HT No : 2303A54015

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 program that takes a number as user input and checks whether it is a palindrome number.” Code:

```
#"Write a Python program that takes a number as user input and checks whether it is a palindrome number."
def is_palindrome_number(num):
    # Convert the number to string to check for palindrome
    str_num = str(num)
    # Check if the string is equal to its reverse
    return str_num == str_num[::-1]
# Take user input
user_input = input("Enter a number: ")
try:
    number = int(user_input)
    if is_palindrome_number(number):
        print(f"{number} is a palindrome number.")
    else:
        print(f"{number} is not a palindrome number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Output:

```
ktop/Ai_assit_coding/Zero-Shot-palindrome.py
Enter a number: 121
121 is a palindrome number.
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding> & C:/Users/malle/AppData/Local/Programs/Python/Python314/python.exe c:/Users/malle/OneDrive/De
ktop/Ai_assit_coding/Zero-Shot-palindrome.py
Ent
123  Focus folder in explorer (ctrl + click)
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding>
```

Analysis:

1. The zero-shot prompt generated correct basic logic for palindrome checking.
2. The initial solution lacked proper handling for negative and invalid inputs.

3. Additional validation was required to make the program more reliable.

Question 2: One-Shot Prompting (Factorial Calculation) Write a oneshot 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 program to compute the factorial of a number taken as user input.

Code :

```python
#"Write a Python program to compute the factorial of a number taken as user input.
def factorial(n):
    # Base case for factorial
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
# Take user input
user_input = input("Enter a number to compute its factorial: ")
try:
    number = int(user_input)
    if number < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        result = factorial(number)
        print(f"The factorial of {number} is {result}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

OUPUT :

```
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding> & C:/Users/malle/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/malle/OneDrive/De
sktop/Ai_assit_coding/Factorial Calculation.py"
Enter a number to compute its factorial: 5
The factorial of 5 is 120.
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding>
```

Analysis :

1. Providing one example improved the clarity of the generated code.

2. The program correctly handled negative numbers compared to zero-shot prompting.

3. The logic was more structured and easier to understand.

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 program to check whether a number is an Armstrong number using user input.

Code :

Armstrong Number Check.py

```python
#"Write a Python program to check whether a number is an Armstrong number using user input.
def is_armstrong_number(num):
    # Convert the number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)
    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
    # An Armstrong number is one where the sum of the powers equals the original number
    return sum_of_powers == num
# Take user input
user_input = input("Enter a number to check if it is an Armstrong number: ")
try:
    number = int(user_input)
    if is_armstrong_number(number):
        print(f"{number} is an Armstrong number.")
    else:
        print(f"{number} is not an Armstrong number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

OUTPUT :

Analysis :

1. Multiple examples helped the AI identify the Armstrong number pattern accurately.
2. The generated code was well-structured and produced correct results.
3. Few-shot prompting reduced ambiguity compared to zero-shot prompting.

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 e iciency.

• Compare the output with earlier prompting strategies.

Prompt :

#“Write an optimized Python program that takes a number as user input and classifies it as Prime, Composite, or Neither.

Code :

```python
#"Write an optimized Python program that takes a number as user input and classifies it as Prime, Composite, or Neither."
def classify_number(num):
    if num <= 1:
        return "Neither Prime nor Composite"
    elif num == 2:
        return "Prime"
    elif num % 2 == 0:
        return "Composite"
    else:
        for i in range(3, int(num**0.5) + 1, 2):
            if num % i == 0:
                return "Composite"
        return "Prime"
# Take user input
user_input = input("Enter a number to classify as Prime, Composite, or Neither: ")
try:
    number = int(user_input)
    classification = classify_number(number)
    print(f"{number} is classified as: {classification}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

OUTPUT :

```
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding> & C:/Users/malle/AppData/Local/Programs/Python/Python314/python.exe c:/Users/malle/OneDrive/Des
ktop/Ai_assit_coding/Prime.py
Enter a number to classify as Prime, Composite, or Neither: 2
2 is classified as: Prime.
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding> & C:/Users/malle/AppData/Local/Programs/Python/Python314/python.exe c:/Users/malle/OneDrive/Des
ktop/Ai_assit_coding/Prime.py
Enter a number to classify as Prime, Composite, or Neither: 25
25 is classified as: Composite.
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding> & C:/Users/malle/AppData/Local/Programs/Python/Python314/python.exe c:/Users/malle/OneDrive/Des
ktop/Ai_assit_coding/Prime.py
Enter a number to classify as Prime, Composite, or Neither: 13
13 is classified as: Prime.
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding>
```

Analysis :

1. Clear instructions and constraints resulted in an optimized solution.
2. The program e iciently classified numbers using minimal iterations.
3. Proper input validation improved program reliability.

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 ine iciencies in the logic.

Prompt :

#“Write a Python program that takes a number as user input and checks whether it is a perfect number.” Code :

```python
#"Write a Python program that takes a number as user input and checks whether it is a perfect number."
def is_perfect_number(num):
    # A perfect number is equal to the sum of its proper divisors
    if num < 1:
        return False
    sum_of_divisors = sum(i for i in range(1, num) if num % i == 0)
    return sum_of_divisors == num
# Take user input
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("Invalid input. Please enter a valid integer.")
```

OUTPUT :

```
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding> & C:/Users/malle/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/malle/OneDrive/Desktop/Ai_assit_coding/Perfect Number Check.py"
Enter a number to check if it is a perfect number: 6
6 is a perfect number.
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding> & C:/Users/malle/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/malle/OneDrive/Desktop/Ai_assit_coding/Perfect Number Check.py"
Enter a number to check if it is a perfect number: 2
2 is not a perfect number.
PS C:\Users\malle\OneDrive\Desktop\Ai_assit_coding>
```

Analysis :

1. The generated code correctly identified perfect numbers.
2. The solution was ine icient for large inputs due to unnecessary iterations.
3. Edge-case handling and optimization were missing initially.

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 number is even or odd using user input with proper validation.

Code :

OUTPUT :

Analysis :

1. Examples improved input handling and output clarity.
2. The program correctly handled negative numbers and zero.
3. Input validation prevented errors for non-integer values.