Name : N. Soumya

H NO:2303A51869

Batch-13

ASSIGNMENT-3.1

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

Write a zero-shot prompt (without providing any examples) to generate

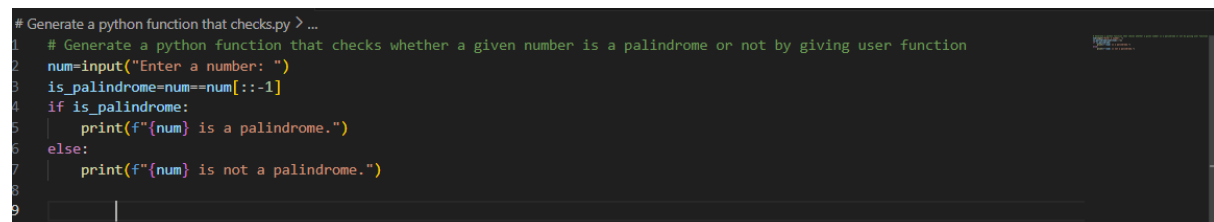a Python function that checks whether a given number is a palindrome.

Task:

• Record the AI-generated code.

• Test the code with multiple inputs.

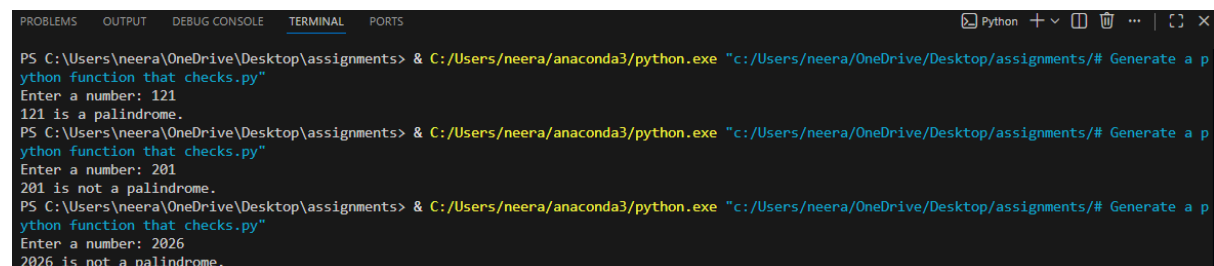• Identify any logical errors or missing edge-case handling.

PROMPT:

Generate a python function that checks whether a given number is a
palindrome or not by giving user function

Code:

```
# Generate a python function that checks whether a given number is a palindrome or not by giving user function
num=input("Enter a number: ")
is_palindrome=num==num[::-1]
if is_palindrome:
    print(f"{num} is a palindrome.")
else:
    print(f"{num} is not a palindrome.")
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/# Generate a p
ython function that checks.py"
Enter a number: 121
121 is a palindrome.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/# Generate a p
ython function that checks.py"
Enter a number: 201
201 is not a palindrome.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/# Generate a p
ython function that checks.py"
Enter a number: 2026
2026 is not a palindrome.
```

Justification:

A palindrome number is a number that remains the same when read forward
and backward.
In this program, the number is taken as input from the user .The input is

treated as a string and reversed using Python slicing [::-1].The original number is compared with its reversed form .If both are equal, the number is identified as a palindrome; otherwise, it is not.

#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.
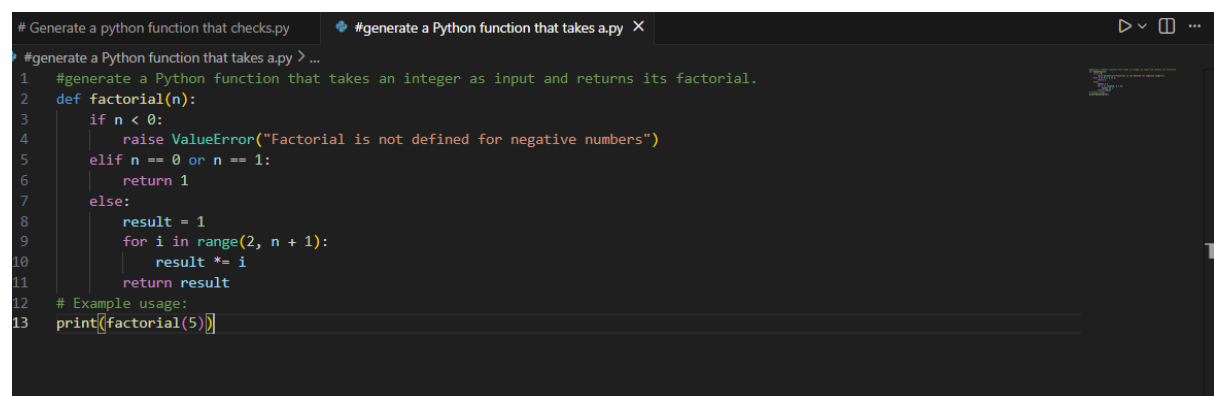
Example:

Input: 5 → Output: 120

Task:

• Compare the generated code with a zero-shot solution.

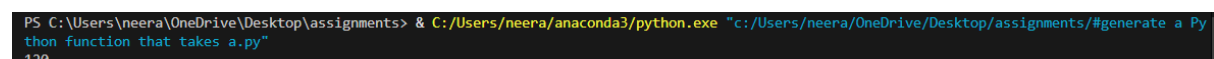• Examine improvements in clarity and correctness.

Prompt:

#Generate a Python function that takes an integer as input and returns its factorial.

Code:

```python
#generate a Python function that takes an integer as input and returns its factorial.
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Example usage:
print(factorial(5))
```

Output:

```
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#generate a Python function that takes a.py"
120
```

Justification:

The function factorial(n) computes the factorial of a given number using an iterative approach. It first checks if the input is negative and raises an error since factorial is undefined for negative numbers .For base cases 0 and 1, the function returns 1 as their factorial value .For other positive integers, a loop multiplies numbers from 2 to n to compute the factorial.

This method is efficient, easy to understand, and avoids recursion overhead**.**

#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 takes an integer as input and checks whether a given number is an Armstrong number or not

CODE:

```
Untitled-1.py > ...
1    #Generate a Python function that takes an integer as input andchecks whether a given number is an Armstrong number or not
2    def is_armstrong(n):
3        if n < 0:
4            return "Invalid input"
5        digits = str(n)
6        power = len(digits)
7        total = 0
8
9        for d in digits:
10           total += int(d) ** power
11
12       if total == n:
13           return "Armstrong Number"
14       else:
15           return "Not an Armstrong Number"
16
17   # Example usages:
18   print("153 →", is_armstrong(153))
19   print("370 →", is_armstrong(370))
20   print("123 →", is_armstrong(123))
21   print("0 →", is_armstrong(0))
22   print("1 →", is_armstrong(1))
23
```

OUTPUT:

```
0: True
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe c:/Users/neera/OneDrive/Desktop/assignments/Untitled-1.py
153 → Armstrong Number
370 → Armstrong Number
123 → Not an Armstrong Number
0 → Armstrong Number
1 → Armstrong Number
PS C:\Users\neera\OneDrive\Desktop\assignments> ▯
```

JUSTIFICATION:

The function is_ Armstrong (n) checks whether a given number satisfies the Armstrong number property.

It first validates the input by rejecting negative numbers as invalid.

The digits of the number are extracted and each digit is raised to the power of the total number of digits.

The sum of these powered digits is compared with the original number.

If both values match, the number is an Armstrong number; otherwise, it is not.

#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:

Write an optimized Python program that asks the user to enter an integer and classifies the number as Prime, Composite, or Neither.

CODE:

```python
#Write an optimized Python program that asks the user to enter an integer and classifies the number as Prime, Composite, or
def classify_number(n):
    if n <= 1:
        return "Neither"
    elif n == 2:
        return "Prime"
    elif n % 2 == 0:
        return "Composite"

    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return "Composite"

    return "Prime"
def main():
    try:
        number = int(input("Enter an integer: "))
        classification = classify_number(number)
        print(f"The number {number} is classified as: {classification}")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe c:/Users/neera/OneDrive/Desktop/assignments/Untitled-2.py
Enter an integer: 123
The number 123 is classified as: Composite
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe c:/Users/neera/OneDrive/Desktop/assignments/Untitled-2.py
Enter an integer: 27
The number 27 is classified as: Composite
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe c:/Users/neera/OneDrive/Desktop/assignments/Untitled-2.py
Enter an integer: 7
The number 7 is classified as: Prime
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe c:/Users/neera/OneDrive/Desktop/assignments/Untitled-2.py
Enter an integer: 17
The number 17 is classified as: Prime
```

JUSTIFICATION:

The program correctly classifies a number as Prime, Composite, or Neither with proper input validation to avoid errors. Special cases such as negative numbers, 0, and 1 are handled separately, as they are neither prime nor composite.

The prime-checking logic is optimized by testing divisibility only up to the square root of the number, reducing time complexity to O($\sqrt{n}$). The use of a function and clear comments improves readability and makes the code easy to understand and compare with earlier prompting strategies.

#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 function to check whether a given positive integer is a perfect number. The function should return True or False.

CODE:

```python
#Write a Python function to check whether a given positive integer is a perfect number. The function should return True or False
def is_perfect_number(n):
    if n <= 0:
        return False

    divisors_sum = sum(i for i in range(1, n) if n % i == 0)

    return divisors_sum == n
# Example usage:
print(is_perfect_number(6))
print(is_perfect_number(28))
print(is_perfect_number(12))
```

OUTPUT:

```
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python function to check whethe.py"
True
True
False
PS C:\Users\neera\OneDrive\Desktop\assignments> []
```

JUSTIFICATION:

The program validates the input to ensure only positive integers are processed, as perfect numbers are defined only for positive values. The logic efficiently calculates the sum of proper divisors instead of checking all numbers up to the given value. Clear comments and a modular function improve readability, while optimized divisor checking reduces unnecessary computations.

#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 asks the user to enter a number and determines whether it is Even or Odd, with proper input validation.

CODE:

```python
#Write a Python program that asks the user to enter a number and determines whether it is Even or Odd, with proper input validat
def is_even_or_odd():
    while True:
        user_input = input("Please enter a number: ")
        try:
            number = int(user_input)
            if number % 2 == 0:
                print(f"The number {number} is Even.")
            else:
                print(f"The number {number} is Odd.")
            break
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
if __name__ == "__main__":
    is_even_or_odd()
```

OUTPUT:

```
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 17
The number 17 is Odd.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 27
The number 27 is Odd.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 15
The number 15 is Odd.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 06
The number 6 is Even.
```

JUSTIFICATION:

The provided examples guide the AI to correctly classify numbers as even or odd, including the edge case of zero. Input validation ensures non-integer values are handled gracefully, improving robustness. The few-shot approach improves output clarity and consistency compared to zero-shot prompting, especially when handling edge cases such as negative numbers.