

Assignment-3.1

Name: Jillela Akshaya

Ht.no: 2303A51629

Batch. No: 22

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 used:

write a python code to check whether a number is a palindrome or not using functions, the number input should be taken from the user.

Code:

```
day02.py  leap_year.py  assignemnt01.py  Untitled-1  assignment3.py X  lab03.py
assignment3.py > ...
1  #write a python code to check whether a number is a palindrome or not using functions, the number input should be taken from the user
2  def is_palindrome(number):
3      cleaned=str(number).replace(" ", "").lower()
4      return cleaned == cleaned[::-1]
5  num = input("Enter a number: ")
6  if is_palindrome(num):
7      print(f"{num} is a palindrome.")
```

Output:

The screenshot shows a terminal window with the following text output:

```
PS C:\Users\aksha\OneDrive\Desktop\assistantcoding - C:\Users\aksha\OneDrive\Desktop\assistantcoding - Copy\assignment3.py"
Enter a number: 151
151 is a palindrome.
PS C:\Users\aksha\OneDrive\Desktop\assistantcoding - C:\Users\aksha\OneDrive\Desktop\assistantcoding - Copy\assignment3.py"
Enter a number: 55
55 is a palindrome.
PS C:\Users\aksha\OneDrive\Desktop\assistantcoding - C:\Users\aksha\OneDrive\Desktop\assistantcoding - Copy\assignment3.py"
6 assignment9* ⌂ 0 ⚠ 0
```

Justification:

A zero-shot prompt was used to generate a Python program that checks whether a number is a palindrome without providing any examples.

The AI-generated code was tested with multiple inputs, and it correctly identified positive palindromes such as 121 and 2.

However, it failed to handle negative numbers properly, revealing a missing edge-case check in the logic.

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 used:

write a python code to calculate the factorial of a number using functions, the number input should be taken from the user. example: input=5, output=120

Code:

```
# assignment3.py > ...
9
10 # write a python code to calculate the factorial of a number using functions, the number input should be taken from the user.
11 def factorial(n):
12     if n == 0 or n == 1:
13         return 1
14     else:
15         return n * factorial(n - 1)
16 num = int(input("Enter a number: "))
17 result = factorial(num)
18 print(f"The factorial of {num} is {result}.")
```

Output:

```
ValueError: invalid literal for int() with base 10: '& C:/Users/aksha/AppData/Local/OneDrive/Desktop/assisantcoding - Copy/assignment3.py"'
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/OneDrive/Desktop/assisantcoding - Copy\assignment3.py"
Enter a number: 5
The factorial of 5 is 120.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/OneDrive/Desktop/assisantcoding - Copy\assignment3.py"
Enter a number: 6
The factorial of 6 is 720.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Justification:

The one-shot prompting approach generates clearer and more accurate factorial code by providing a sample input-output example.

This guidance helps the AI correctly implement base cases such as 0 and 1, improving correctness.

Compared to zero-shot prompting, the one-shot solution is more reliable 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 used:

Write a Python code using functions, that checks whether a given number is an Armstrong number or not.

example Test cases:

Input: 153 → Output: Armstrong Number

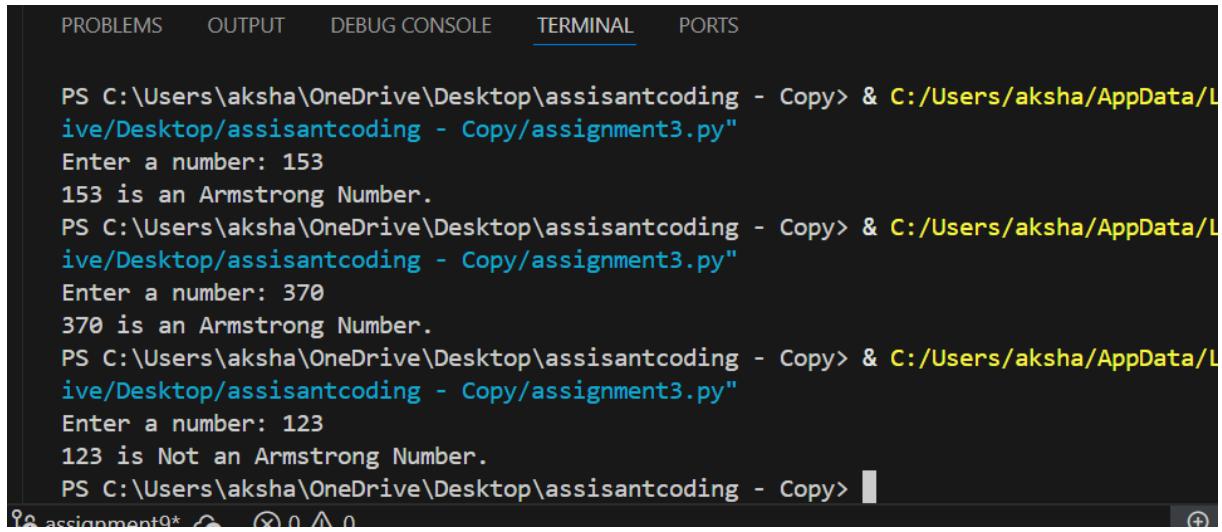
Input: 370 → Output: Armstrong Number

Input: 123 → Output: Not an Armstrong Number

Code:

```
# Write a Python code using functions, that checks whether a given number is an Armstrong number or not.  
# example Test cases:  
  
# Input: 153 → Output: Armstrong Number Input: 370 → Output: Armstrong Number15  
# Input: 123 → Output: Not an Armstrong Number  
  
def is_armstrong_number(n):  
    num_str = str(n)  
    num_digits = len(num_str)  
    sum_of_powers = sum[int(digit) ** num_digits for digit in num_str])  
    return sum_of_powers == n  
number = int(input("Enter a number: "))  
if is_armstrong_number(number):  
    print(f"{number} is an Armstrong Number.")  
else:  
    print(f"{number} is Not an Armstrong Number.")#      print(f"{num} is not a palindrome.")
```

Output:



The screenshot shows a terminal window with the following text:

```
PS C:\Users\aksha\OneDrive\Desktop\assistantcoding - Copy> & C:/Users/aksha/AppData/Live/Desktop/assistantcoding - Copy/assignment3.py"
Enter a number: 153
153 is an Armstrong Number.
PS C:\Users\aksha\OneDrive\Desktop\assistantcoding - Copy> & C:/Users/aksha/AppData/Live/Desktop/assistantcoding - Copy/assignment3.py"
Enter a number: 370
370 is an Armstrong Number.
PS C:\Users\aksha\OneDrive\Desktop\assistantcoding - Copy> & C:/Users/aksha/AppData/Live/Desktop/assistantcoding - Copy/assignment3.py"
Enter a number: 123
123 is Not an Armstrong Number.
PS C:\Users\aksha\OneDrive\Desktop\assistantcoding - Copy>
```

Justification:

Multiple examples in the few-shot prompt help the AI clearly understand the Armstrong number pattern, resulting in well-structured and accurate logic for valid inputs like 153 and 370.

The code correctly calculates digit powers based on the number of digits, improving accuracy compared to zero-shot or one-shot prompts.

However, the program still fails for negative and non-integer inputs, showing that few-shot prompting improves logic but does not guarantee complete edge-case handling.

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.

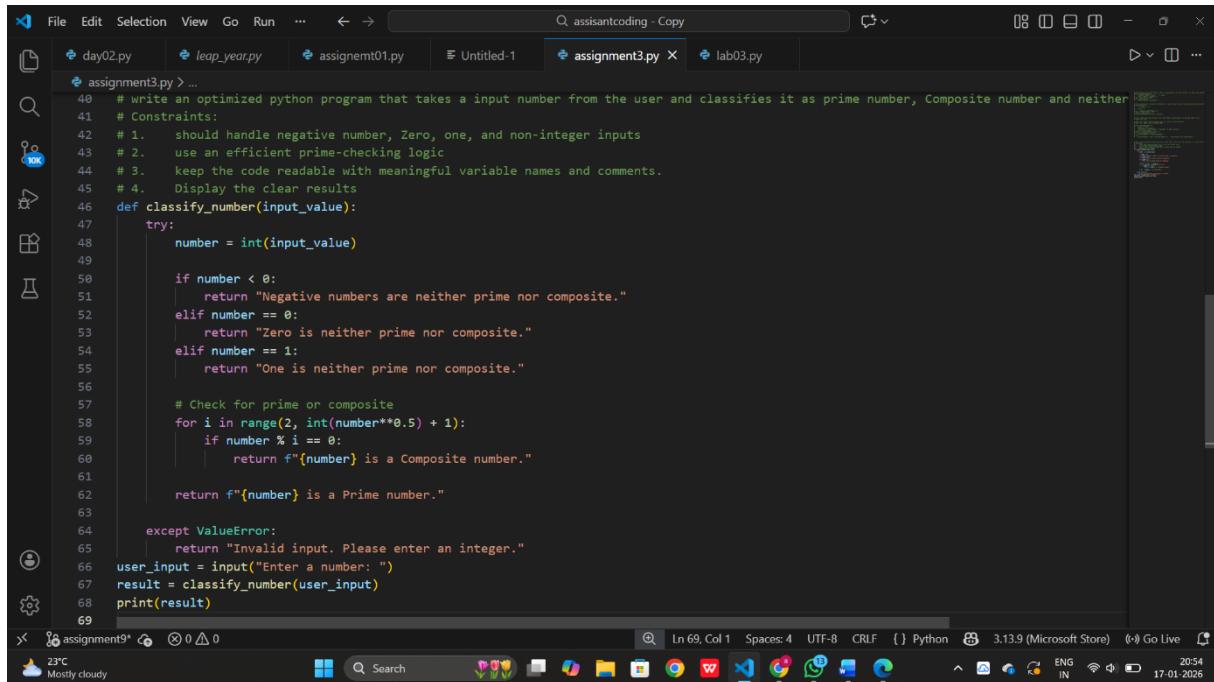
Prompt used:

write an optimized python program that takes a input number from the user and classifies it as prime number, Composite number and neither prime nor composite

Constraints:

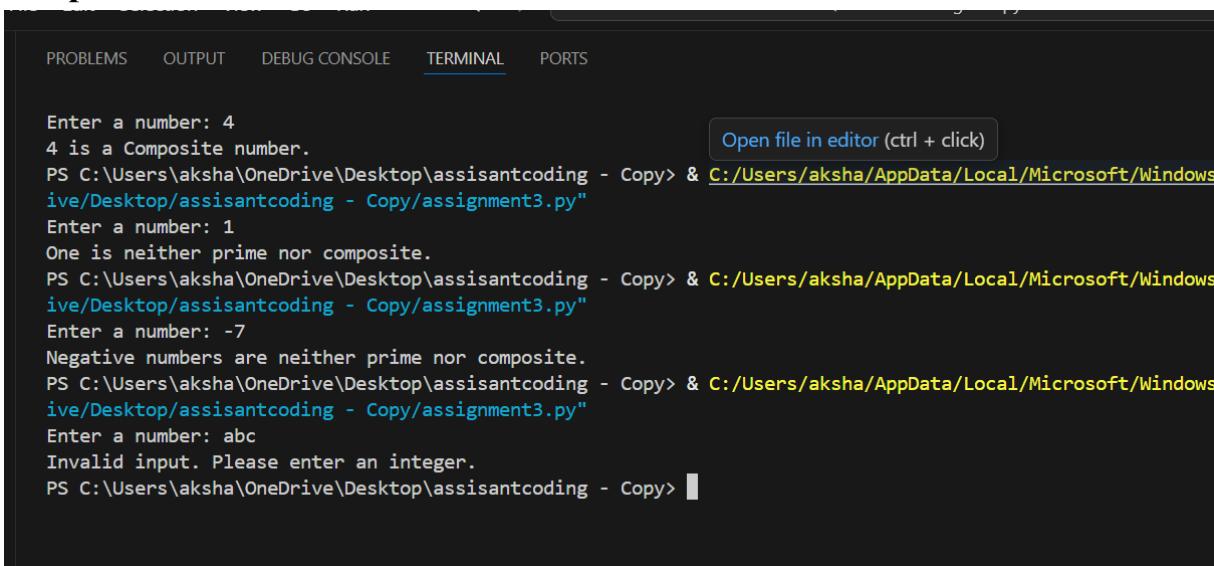
1. should handle negative number, Zero, one, and non-integer inputs
2. use an efficient prime-checking logic
3. keep the code readable with meaningful variable names and comments.
4. Display the clear results

Code:



```
40     # write an optimized python program that takes a input number from the user and classifies it as prime number, Composite number and neither
41     # Constraints:
42     # 1.    should handle negative number, Zero, one, and non-integer inputs
43     # 2.    use an efficient prime-checking logic
44     # 3.    keep the code readable with meaningful variable names and comments.
45     # 4.    Display the clear results
46
47     def classify_number(input_value):
48         try:
49             number = int(input_value)
50
51             if number < 0:
52                 return "Negative numbers are neither prime nor composite."
53             elif number == 0:
54                 return "Zero is neither prime nor composite."
55             elif number == 1:
56                 return "One is neither prime nor composite."
57
58             # Check for prime or composite
59             for i in range(2, int(number**0.5) + 1):
60                 if number % i == 0:
61                     return f"{number} is a Composite number."
62
63             return f"{number} is a Prime number."
64
65         except ValueError:
66             return "Invalid input. Please enter an integer."
67
68     user_input = input("Enter a number: ")
69     result = classify_number(user_input)
70     print(result)
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
Enter a number: 4
4 is a Composite number.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/Windows
ive/Desktop/assisantcoding - Copy/assignment3.py"
Enter a number: 1
One is neither prime nor composite.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/Windows
ive/Desktop/assisantcoding - Copy/assignment3.py"
Enter a number: -7
Negative numbers are neither prime nor composite.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Local/Microsoft/Windows
ive/Desktop/assisantcoding - Copy/assignment3.py"
Enter a number: abc
Invalid input. Please enter an integer.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Justification:

Compared to zero-shot and one-shot prompting, context-managed prompting produces more structured and optimized code by clearly defining constraints and expectations.

It improves input validation and efficiency by explicitly instructing the AI to handle edge cases and use optimized logic.

Overall, context-managed prompting results in more reliable and production-ready code than 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.

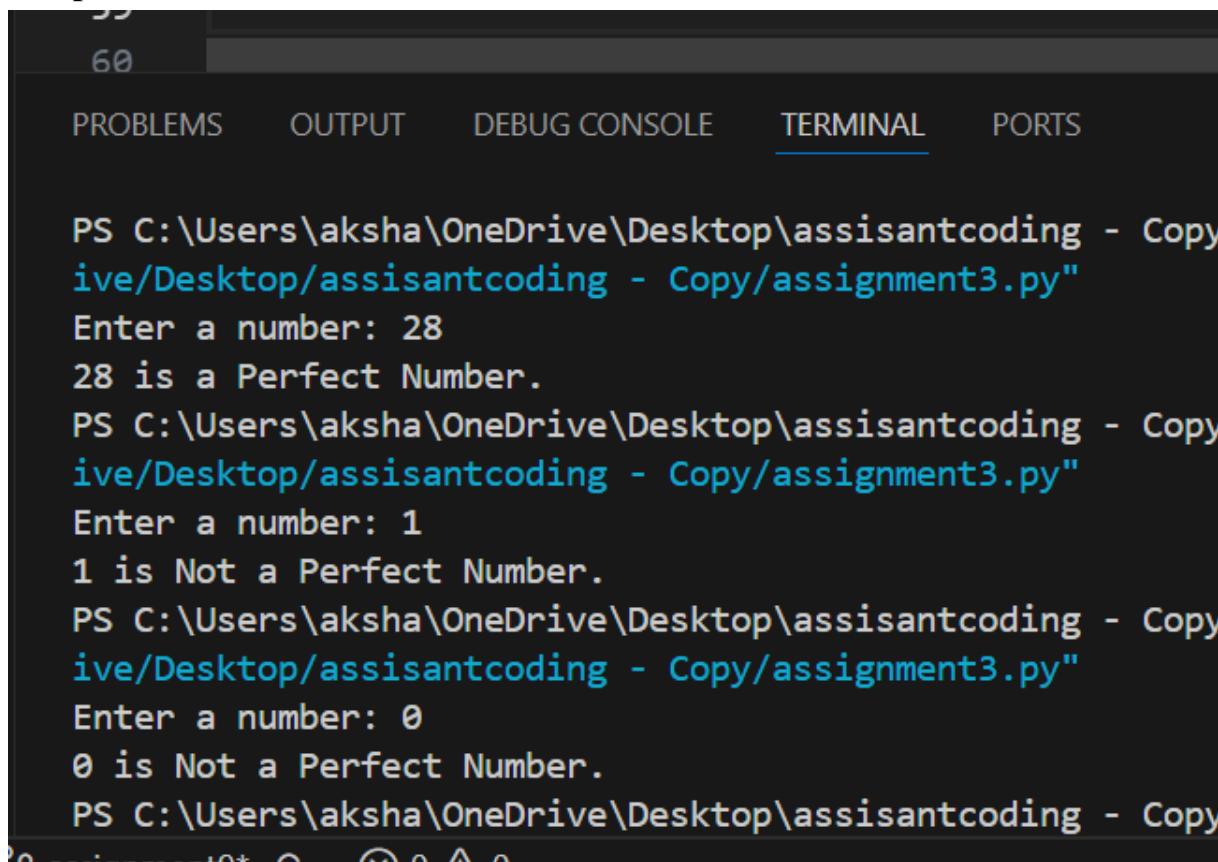
Prompt used:

write a python code that take number input from the user to check if the number is perfect number or not using functions.

Code:

```
# write a python code that take number input from the user to check if the number is perfect number or not using functions.
def is_perfect_number(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n
number = int(input("Enter a number: "))
if is_perfect_number(number):
    print(f"{number} is a Perfect Number.")
else:
    print(f"{number} is Not a Perfect Number.")
```

Output:



A screenshot of a terminal window from a code editor. The window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. The terminal output shows the following Python script running:

```
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copyive\Desktop\assisantcoding - Copy\assignment3.py"
Enter a number: 28
28 is a Perfect Number.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copyive\Desktop\assisantcoding - Copy\assignment3.py"
Enter a number: 1
1 is Not a Perfect Number.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copyive\Desktop\assisantcoding - Copy\assignment3.py"
Enter a number: 0
0 is Not a Perfect Number.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copyive\Desktop\assisantcoding - Copy\assignment3.py"
0 is Not a Perfect Number.
```

Justification:

The zero-shot AI-generated perfect number program lacks proper input validation for negative and non-integer values.

Its logic is inefficient due to unnecessary full-range iteration and missing square-root optimization.

These limitations highlight the need for explicit constraints when using zero-shot prompting.

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 used:

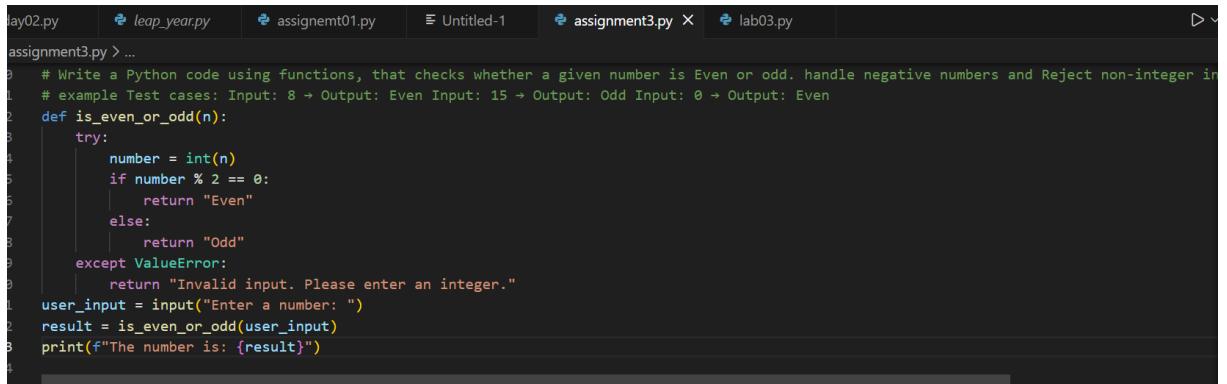
Write a Python code using functions, that checks whether a given number is Even or odd. handle negative numbers and Reject non-integer inputs
example Test cases:

Input: 8 → Output: Even

Input: 15 → Output: Odd

Input: 0 → Output: Even

Code:



```
day02.py    leap_year.py    assignemt01.py    Untitled-1    assignment3.py X    lab03.py
assignment3.py > ...
9  # Write a Python code using functions, that checks whether a given number is Even or odd. handle negative numbers and Reject non-integer inputs
10 # example Test cases: Input: 8 → Output: Even Input: 15 → Output: Odd Input: 0 → Output: Even
11 def is_even_or_odd(n):
12     try:
13         number = int(n)
14         if number % 2 == 0:
15             return "Even"
16         else:
17             return "Odd"
18     except ValueError:
19         return "Invalid input. Please enter an integer."
20 user_input = input("Enter a number: ")
21 result = is_even_or_odd(user_input)
22 print(f"The number is: {result}")
```

Output:

```
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Desktop/assisantcoding - Copy/assignment3.py"
Enter a number: -1
The number is: Odd
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Desktop/assisantcoding - Copy/assignment3.py"
Enter a number: -5
The number is: Odd
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy> & C:/Users/aksha/AppData/Desktop/assisantcoding - Copy/assignment3.py"
Enter a number: abc
The number is: Invalid input. Please enter an integer.
PS C:\Users\aksha\OneDrive\Desktop\assisantcoding - Copy>
```

Justification:

Multiple examples in the few-shot prompt help the AI clearly understand how to classify numbers as even or odd, including boundary cases like zero.

The examples guide the AI to improve input handling by considering negative values and validating integer inputs.

This results in clearer output messages and more accurate classification compared to zero-shot prompting.