Assignment-1

NAME: Hari Charan Gupta

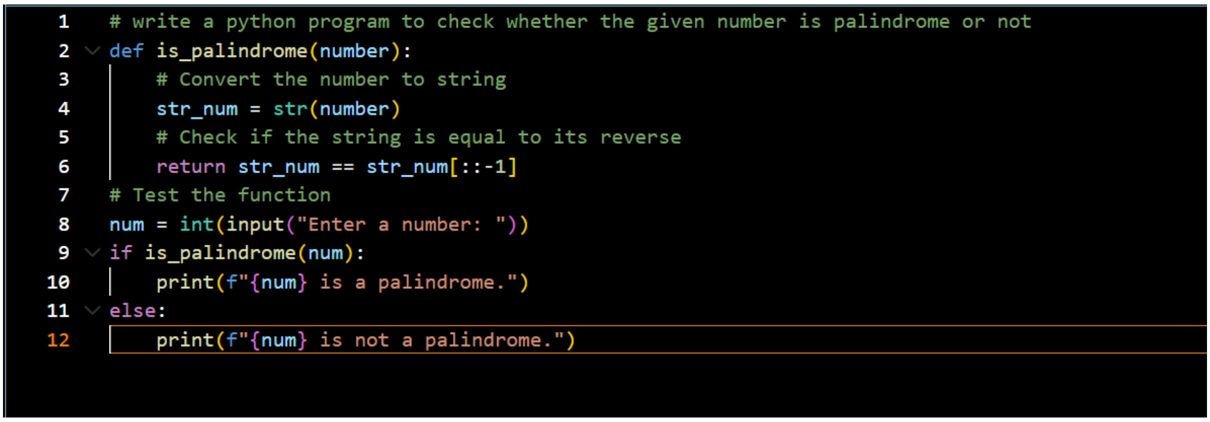
ROLL NO: 2303A52427

COURSE: AI Assisted Coding

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:

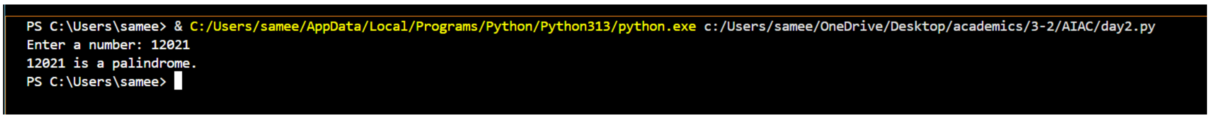
- Record the AI-generated code.
- Test the code with multiple inputs.
- Identify any logical errors or missing edge-case handling.

PROMPT



```python
# write a python program to check whether the given number is palindrome or not
def is_palindrome(number):
    # Convert the number to string
    str_num = str(number)
    # Check if the string is equal to its reverse
    return str_num == str_num[::-1]
# Test the function
num = int(input("Enter a number: "))
if is_palindrome(num):
    print(f"{num} is a palindrome.")
else:
    print(f"{num} is not a palindrome.")
```

OUTPUT



```
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe c:/Users/samee/OneDrive/Desktop/academics/3-2/AIAC/day2.py
Enter a number: 12021
12021 is a palindrome.
PS C:\Users\samee> █
```

ANALYSISI

- The zero-shot prompt gave a correct program for checking palindrome numbers.
- The program works well for normal positive numbers.
- It does not clearly handle negative numbers or wrong input types.
- This shows that the code works, but some special cases are missing.

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.

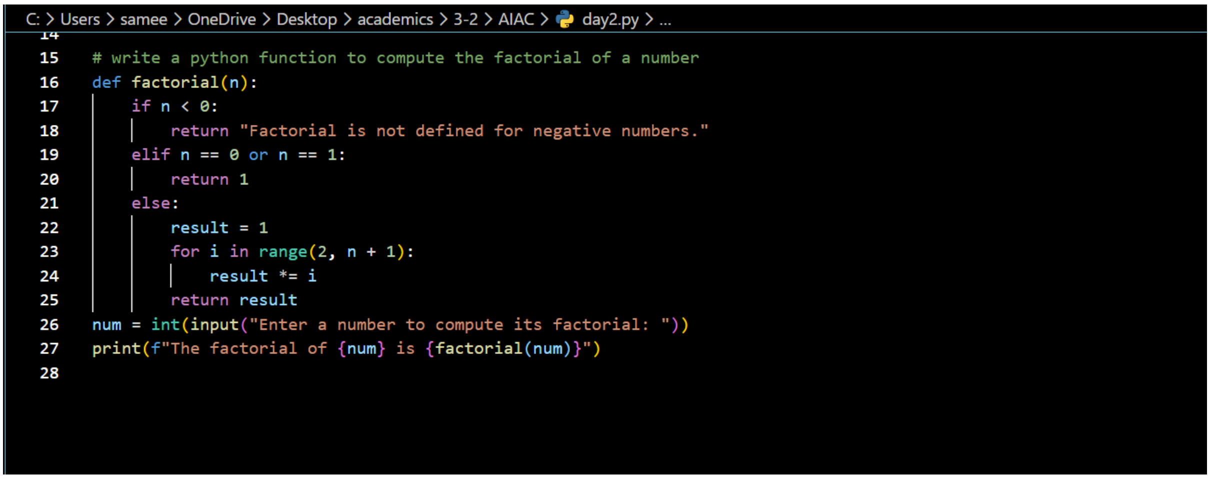
Example:

Input: 5 → Output: 120

Task:

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

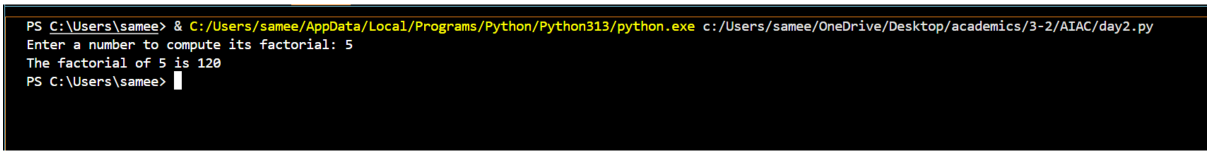
PROMPT



C: > Users > samee > OneDrive > Desktop > academics > 3-2 > AIAC > day2.py > ...

```python
# write a python function to compute the factorial of a number
def factorial(n):
    if n < 0:
        return "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
num = int(input("Enter a number to compute its factorial: "))
print(f"The factorial of {num} is {factorial(num)}")
```

OUTPUT



```
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe c:/Users/samee/OneDrive/Desktop/academics/3-2/AIAC/day2.py
Enter a number to compute its factorial: 5
The factorial of 5 is 120
PS C:\Users\samee> 
```

ANALYSIS

- One-shot prompting produces better and safer code than zero-shot prompting.
- The example helps the AI write clearer logic and handle edge cases correctly.

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

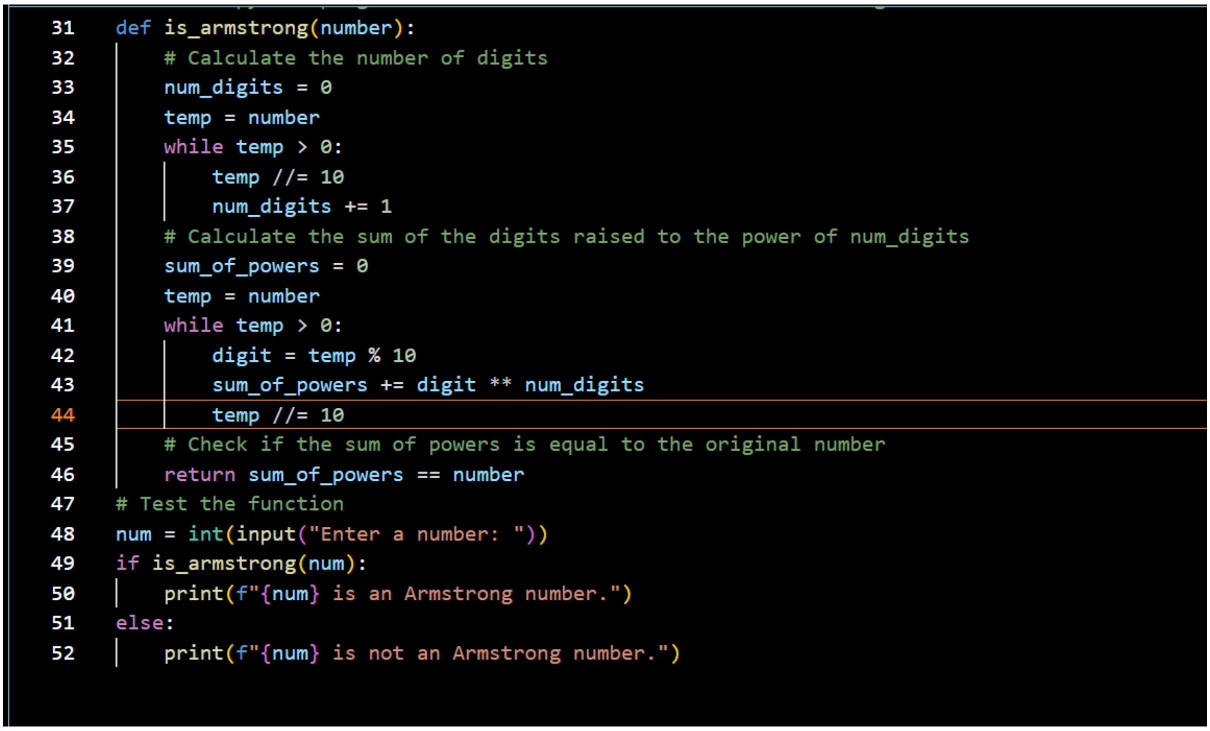
Task:

- Analyze how multiple examples influence code structure and

accuracy.

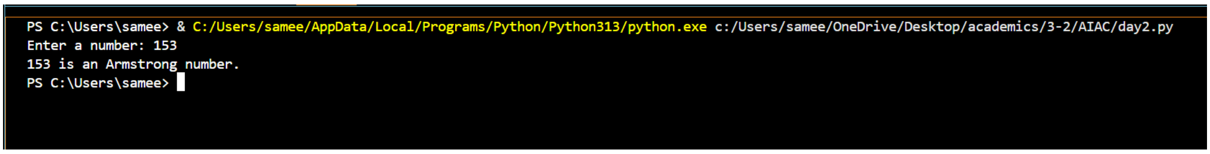
- Test the function with boundary values and invalid inputs.

PROMPT



```python
def is_armstrong(number):
    # Calculate the number of digits
    num_digits = 0
    temp = number
    while temp > 0:
        temp //= 10
        num_digits += 1
    # Calculate the sum of the digits raised to the power of num_digits
    sum_of_powers = 0
    temp = number
    while temp > 0:
        digit = temp % 10
        sum_of_powers += digit ** num_digits
        temp //= 10
    # Check if the sum of powers is equal to the original number
    return sum_of_powers == number
# Test the function
num = int(input("Enter a number: "))
if is_armstrong(num):
    print(f"{num} is an Armstrong number.")
else:
    print(f"{num} is not an Armstrong number.")
```

OUTPUT



```
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe c:/Users/samee/OneDrive/Desktop/academics/3-2/AIAC/day2.py
Enter a number: 153
153 is an Armstrong number.
PS C:\Users\samee> 
```

ANALYSIS

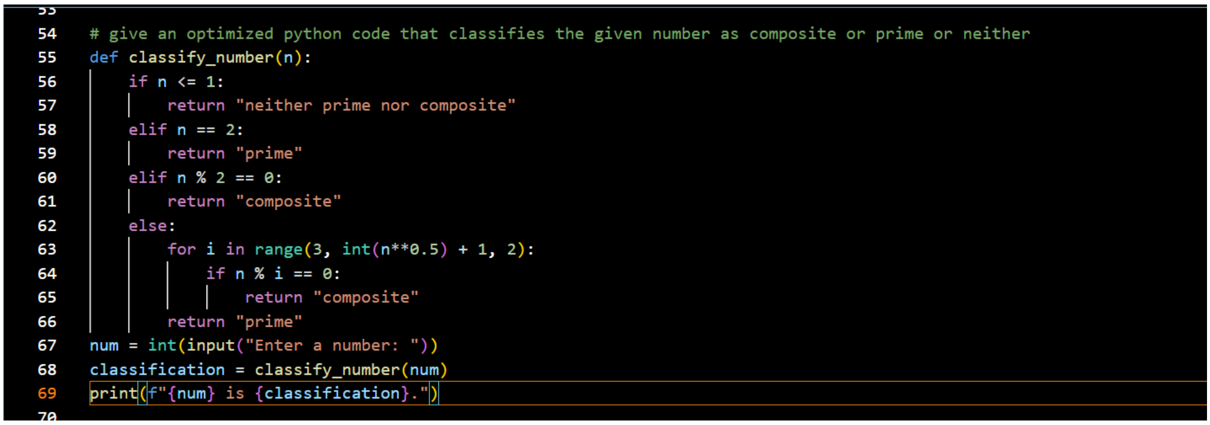
- The code structure is better and follows the examples.
- The code does not handle negative or invalid inputs properly.
- Accuracy is higher compared to zero-shot and one-shot prompting.

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:

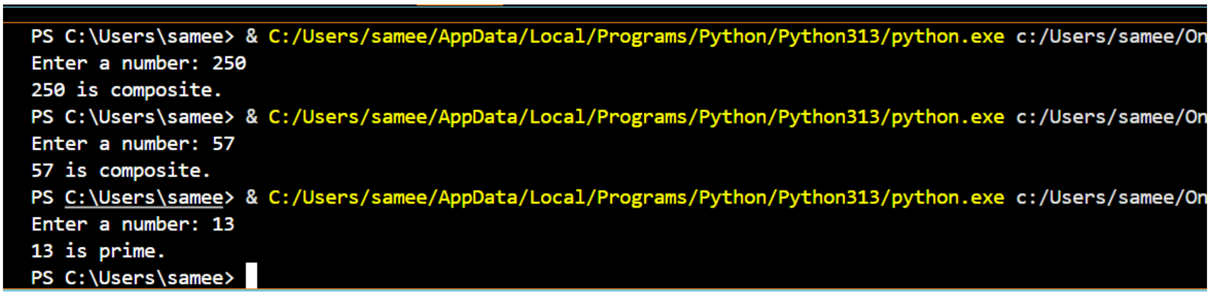
- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

PROMPT



```python
# give an optimized python code that classifies the given number as composite or prime or neither
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"
num = int(input("Enter a number: "))
classification = classify_number(num)
print(f"{num} is {classification}.")
```

OUTPUT



```
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe c:/Users/samee/On
Enter a number: 250
250 is composite.
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe c:/Users/samee/On
Enter a number: 57
57 is composite.
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe c:/Users/samee/On
Enter a number: 13
13 is prime.
PS C:\Users\samee> 
```

ANALYSIS

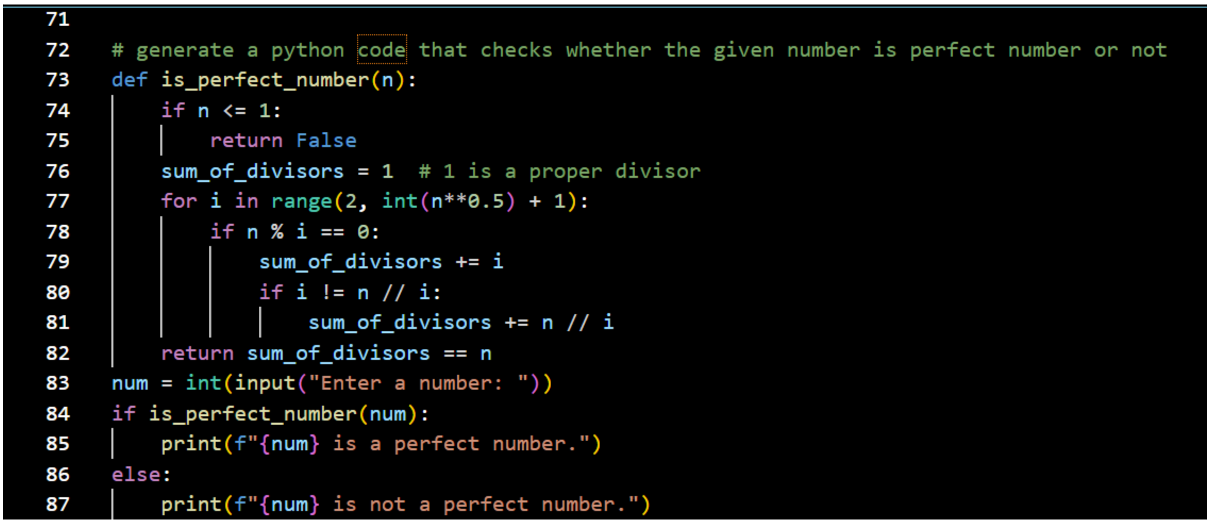
- Input validation is properly handled.
- Compared to earlier prompting methods, this gives the best accuracy and performance.
- Clear instructions helped the AI write correct and efficient code.

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.

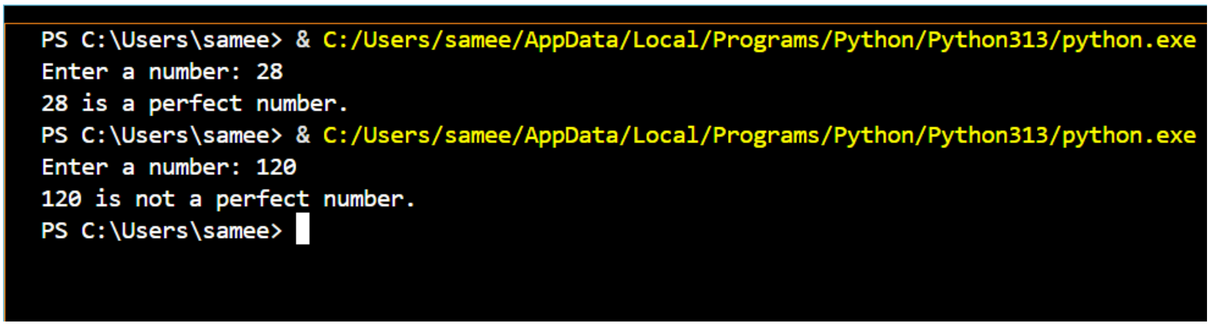
PROMPT



```python

# generate a python code that checks whether the given number is perfect number or not
def is_perfect_number(n):
    if n <= 1:
        return False
    sum_of_divisors = 1  # 1 is a proper divisor
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            sum_of_divisors += i
            if i != n // i:
                sum_of_divisors += n // i
    return sum_of_divisors == n
num = int(input("Enter a number: "))
if is_perfect_number(num):
    print(f"{num} is a perfect number.")
else:
    print(f"{num} is not a perfect number.")
```

OUTPUT



```
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe
Enter a number: 28
28 is a perfect number.
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe
Enter a number: 120
120 is not a perfect number.
PS C:\Users\samee>
```

ANALYSIS

The zero-shot prompt created a correct program for checking perfect numbers.
The program works well for normal positive numbers like 6 and 28.
It does not handle 0, negative numbers, or wrong inputs properly.

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

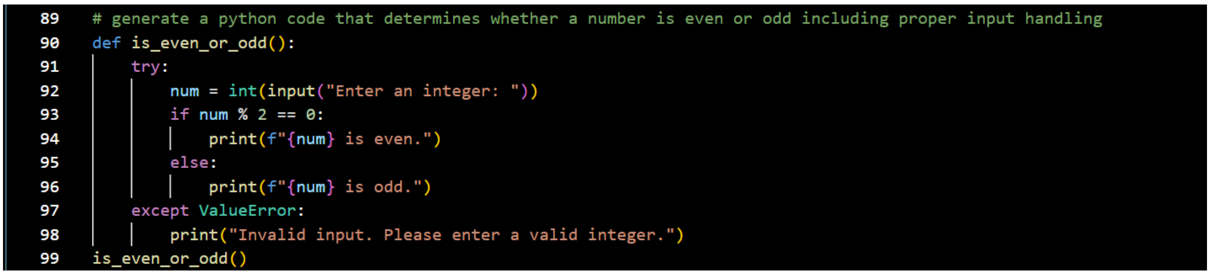
Task:

- Analyze how examples improve input handling and output

clarity.

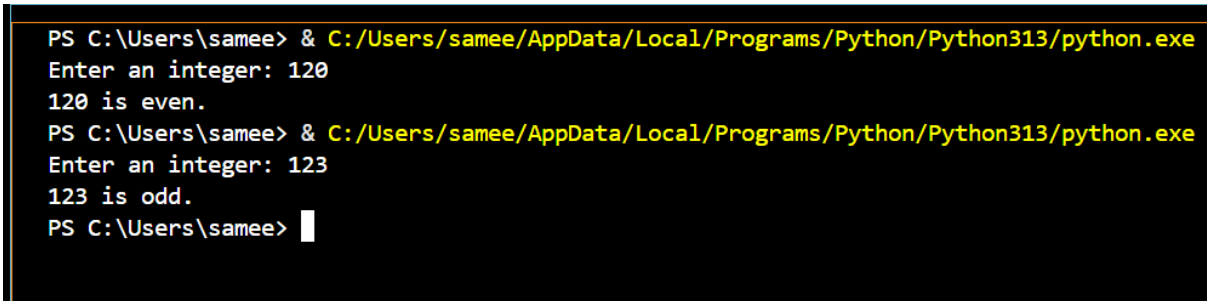
- Test the program with negative numbers and non-integer inputs.

PROMPT



```python
# generate a python code that determines whether a number is even or odd including proper input handling
def is_even_or_odd():
    try:
        num = int(input("Enter an integer: "))
        if num % 2 == 0:
            print(f"{num} is even.")
        else:
            print(f"{num} is odd.")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
is_even_or_odd()
```

OUTPUT



```
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe
Enter an integer: 120
120 is even.
PS C:\Users\samee> & C:/Users/samee/AppData/Local/Programs/Python/Python313/python.exe
Enter an integer: 123
123 is odd.
PS C:\Users\samee>
```

ANALYSIS

The given examples helped the AI understand what even and odd mean.
The program gives clear output like “Even” or “Odd”.
It correctly works for negative numbers also.
It safely rejects non-integer inputs like decimals or strings.