ASSIGNMENT-3.1

Name:M Varshith Reddy

HTNO:2303A52087

Batch:40

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

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

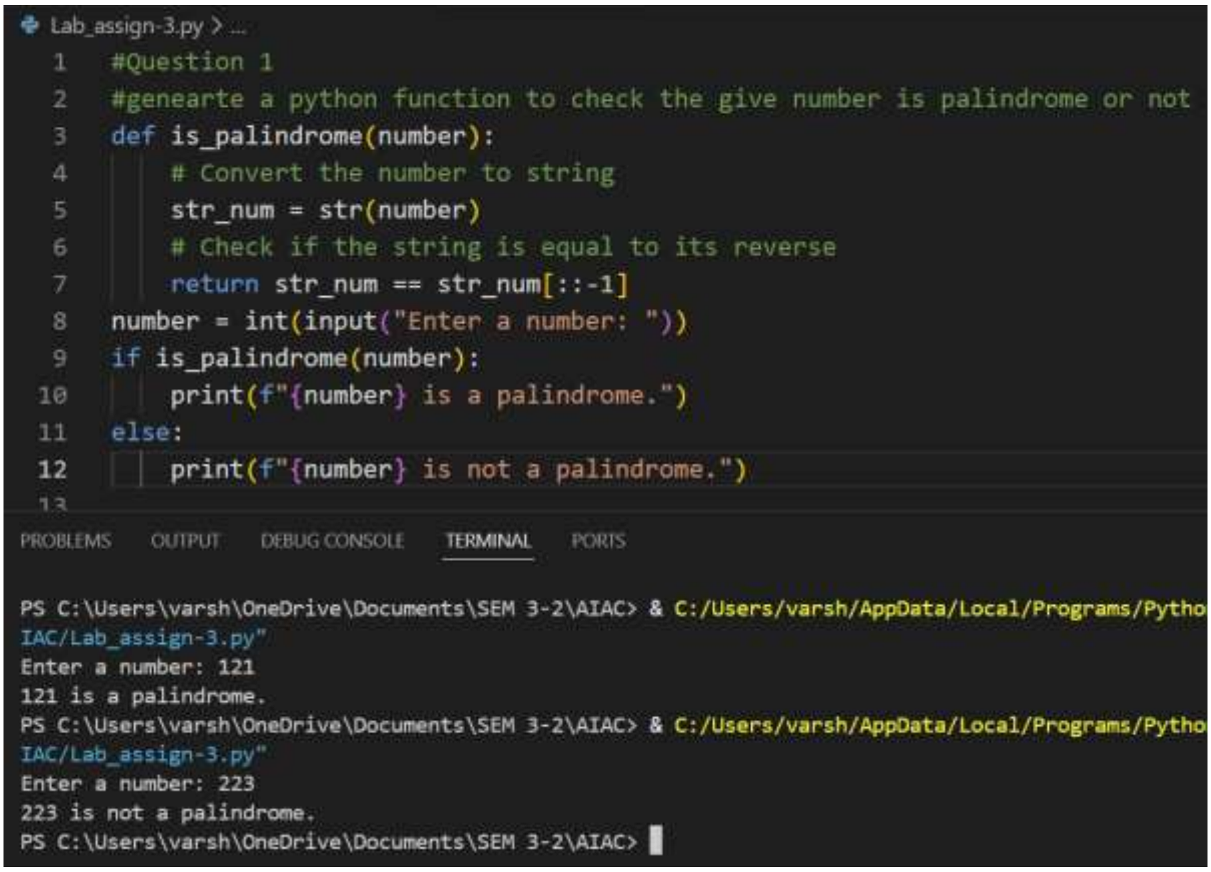
a Python function that checks whether a given number is a palindrome.

Task:

• Record the AI-generated code.

• Test the code with multiple inputs.

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



Lab_assign-3.py > ...

```python
#Question 1
#genearte a python function to check the give 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]
number = int(input("Enter a number: "))
if is_palindrome(number):
    print(f"{number} is a palindrome.")
else:
    print(f"{number} is not a palindrome.")
```

```
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs/Pytho
IAC/Lab_assign-3.py"
Enter a number: 121
121 is a palindrome.
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs/Pytho
IAC/Lab_assign-3.py"
Enter a number: 223
223 is not a palindrome.
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC>
```

Explanation:

Zero-shot prompting relies only on instructions without examples.

It evaluates basic reasoning and may miss edge cases like negatives.

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.

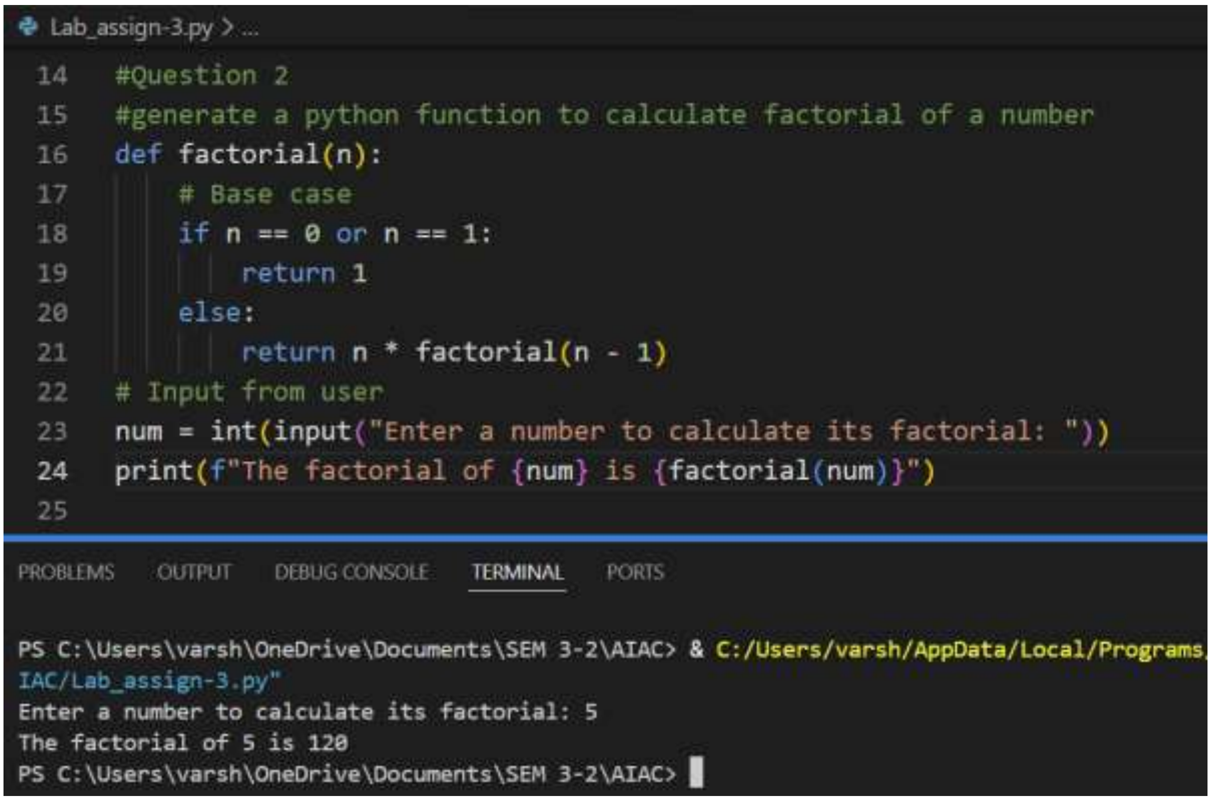
Example:

Input: 5 → Output: 120

Task:

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

• Examine improvements in clarity and correctness.



Lab_assign-3.py > ...

```python
#Question 2
#generate a python function to calculate factorial of a number
def factorial(n):
    # Base case
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
# Input from user
num = int(input("Enter a number to calculate its factorial: "))
print(f"The factorial of {num} is {factorial(num)}")
```

```
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs
IAC/Lab_assign-3.py"
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC>
```

Explanation:

One example helps the AI understand expected logic.

Improves correctness and handling of base cases.

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

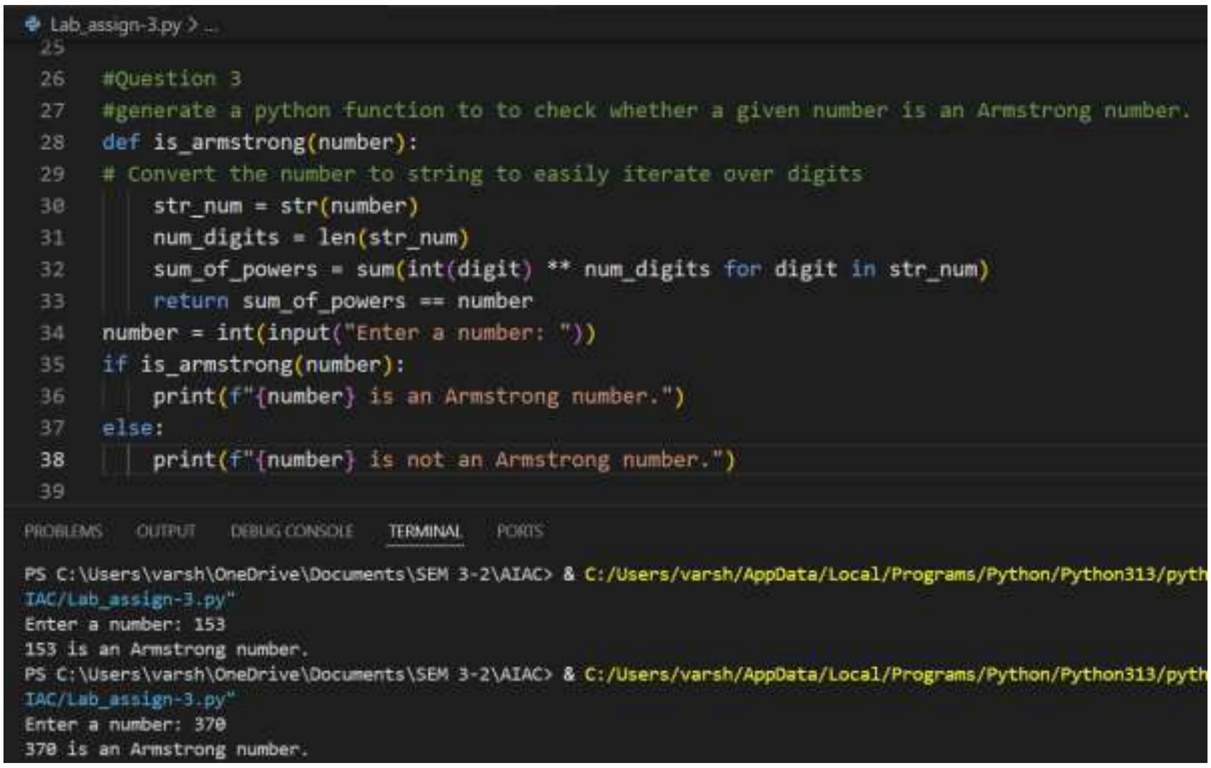
• Input: 123 → Output: Not an Armstrong Number

Task:

• Analyze how multiple examples influence code structure and

accuracy.

• Test the function with boundary values and invalid inputs.



```python
#Question 3
#generate a python function to to check whether a given number is an Armstrong number.
def is_armstrong(number):
# Convert the number to string to easily iterate over digits
    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
number = int(input("Enter a number: "))
if is_armstrong(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

```
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/pyth
IAC/Lab_assign-3.py"
Enter a number: 153
153 is an Armstrong number.
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/pyth
IAC/Lab_assign-3.py"
Enter a number: 370
370 is an Armstrong number.
```

Explanation:

Few-shot prompting uses multiple examples to strongly guide the AI's understanding of the pattern and logic.

This results in improved accuracy, clearer structure, and better handling of boundary and test cases.

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

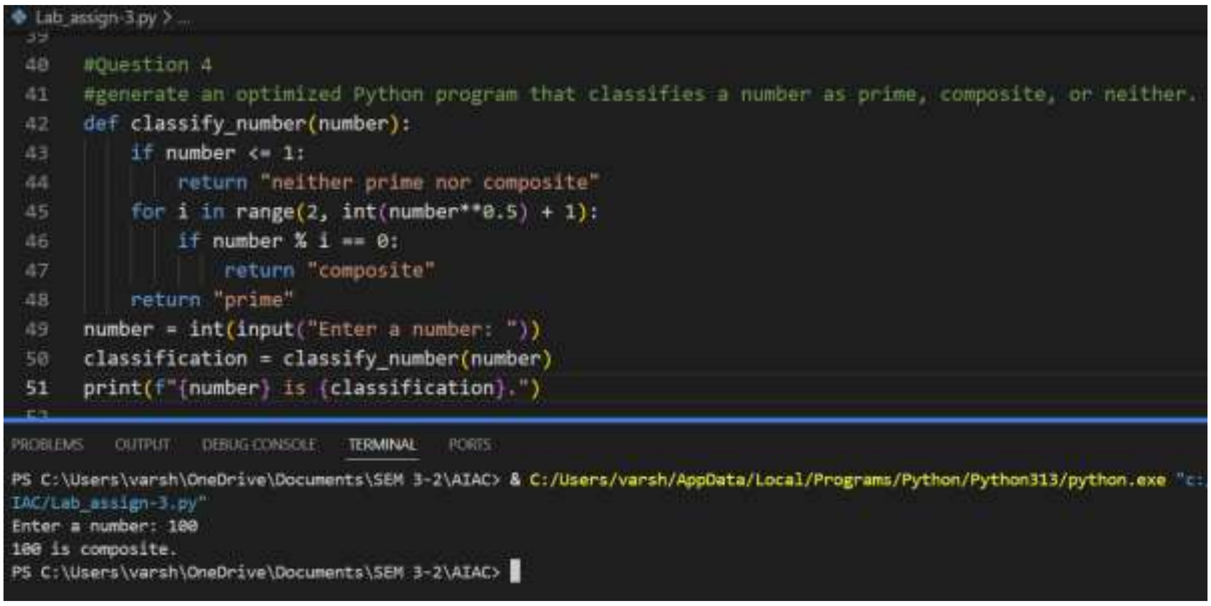
number as prime, composite, or neither.

Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.



```python
#Question 4
#generate an optimized Python program that classifies a number as prime, composite, or neither.
def classify_number(number):
    if number <= 1:
        return "neither prime nor composite"
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return "composite"
    return "prime"
number = int(input("Enter a number: "))
classification = classify_number(number)
print(f"{number} is {classification}.")
```

```
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe "c:
IAC/Lab_assign-3.py"
Enter a number: 100
100 is composite.
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC>
```

Explanation:

Context-managed prompting includes constraints like efficiency and validation, leading to optimized and robust code.

It encourages better design choices compared to basic 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

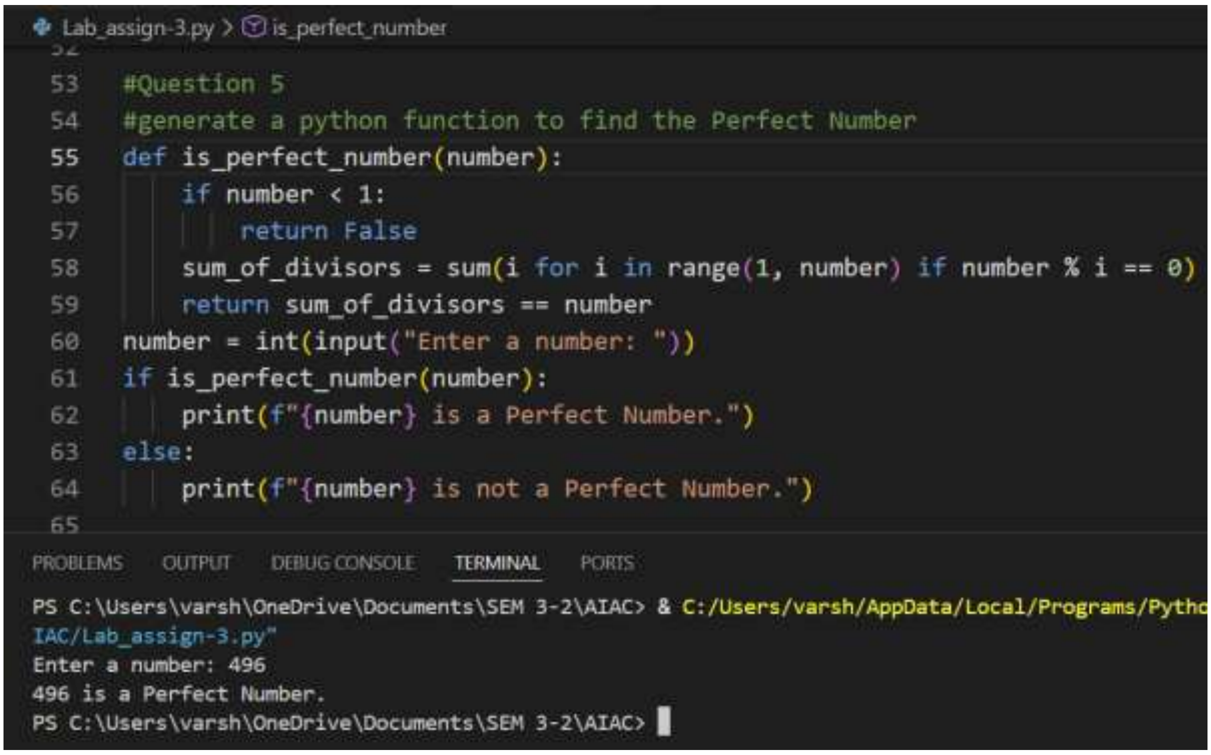
perfect number.

Task:

• Record the AI-generated code.

• Test the program with multiple inputs.

• Identify any missing conditions or inefficiencies in the logic.



Lab_assign-3.py > is_perfect_number

```python
#Question 5
#generate a python function to find the Perfect Number
def is_perfect_number(number):
    if number < 1:
        return False
    sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)
    return sum_of_divisors == number
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.")
```

```
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs/Pytho
IAC/Lab_assign-3.py"
Enter a number: 496
496 is a Perfect Number.
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> 
```

Explanation:

In zero-shot prompting, the AI generates logic without examples, which may lead to inefficiencies or missing checks.

Testing helps identify flaws such as incorrect divisor handling or performance issues.

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

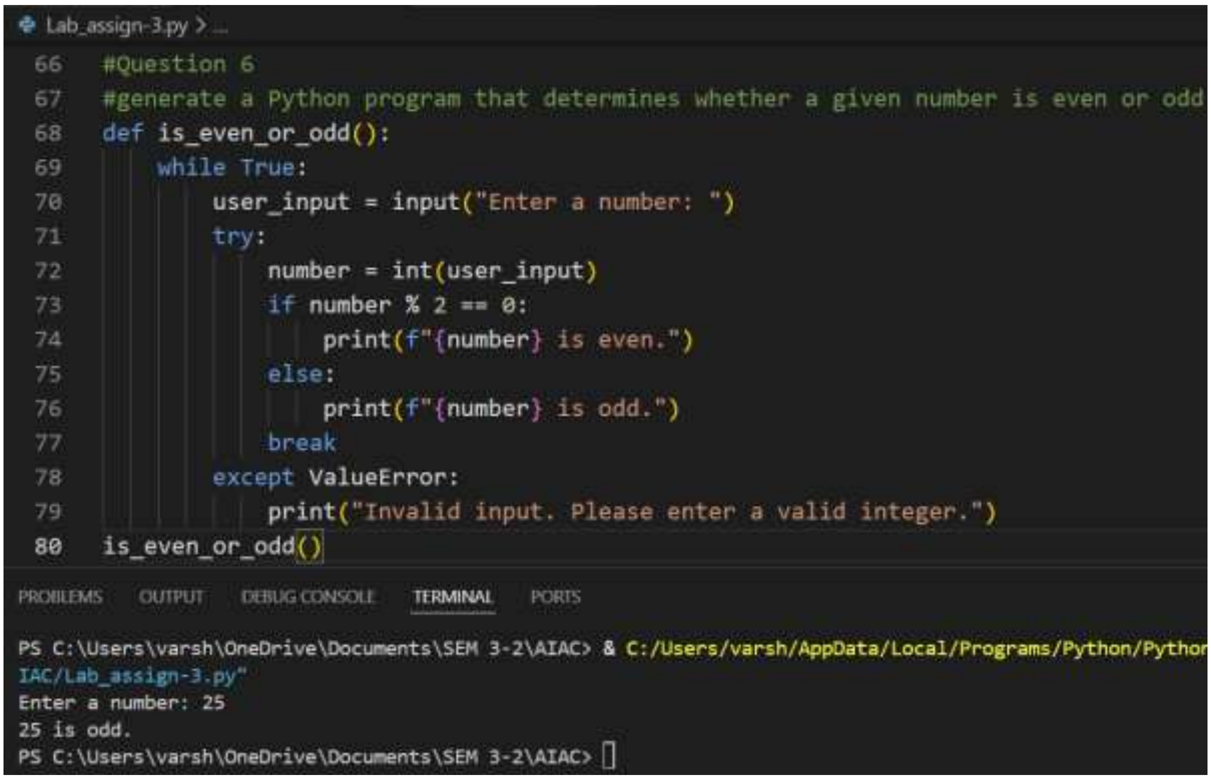
• Input: 0 → Output: Even

Task:

• Analyze how examples improve input handling and output

clarity.

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


Lab_assign-3.py > ...

```python
#Question 6
#generate a Python program that determines whether a given number is even or odd
def is_even_or_odd():
    while True:
        user_input = input("Enter a number: ")
        try:
            number = int(user_input)
            if number % 2 == 0:
                print(f"{number} is even.")
            else:
                print(f"{number} is odd.")
            break
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
is_even_or_odd()
```

```
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Pythor
IAC/Lab_assign-3.py"
Enter a number: 25
25 is odd.
PS C:\Users\varsh\OneDrive\Documents\SEM 3-2\AIAC> 
```

Explanation:

Providing multiple examples improves the AI's understanding of edge cases and input validation.

This results in clearer outputs and better handling of negative and non-integer inputs.