Assignment – 3.1

2303A51176

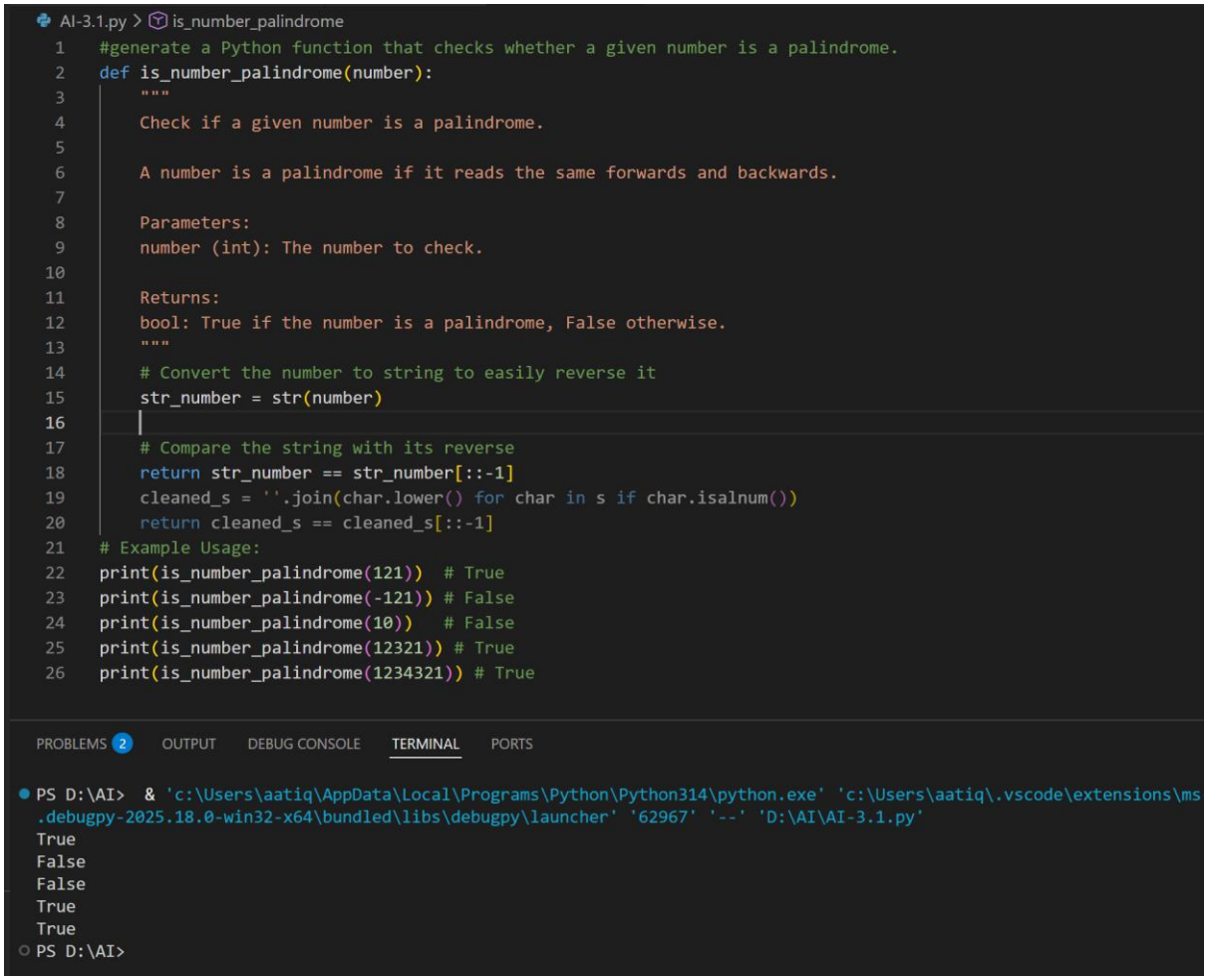
Batch -29

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

PROMPT :

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

CODE and OUTPUT :



Justification :

In zero-shot prompting, the model receives only the problem statement without any supporting examples. This checks the AI's basic understanding of the concept. While generating a palindrome-checking function, the AI typically uses simple logic such as reversing the number or converting it into a string. However, due to the lack of constraints or examples, the solution may ignore important edge cases like negative values, single-digit numbers, or

invalid inputs. This indicates that although zero-shot prompting works well for simple tasks, it may not produce fully robust or real-world-ready code without additional guidance.

Task 2: One-Shot Prompting (Factorial Calculation)

PROMPT :

Generate a Python function to compute the factorial of a given number. Example: Input: 5 → Output: 120.

CODE and OUTPUT :

```python
#generate a Python function to compute the factorial of a given number. Example: Input: 5 → Output: 120.
def compute_factorial(n):
    """
    Compute the factorial of a given number.

    The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.

    Parameters:
    n (int): The number to compute the factorial for.

    Returns:
    int: The factorial of the number.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    elif n == 0 or n == 1:
        return 1
    else:
        factorial = 1
        for i in range(2, n + 1):
            factorial *= i
        return factorial
# Example Usage:
print(compute_factorial(5))  # Output: 120
print(compute_factorial(0))  # Output: 1
print(compute_factorial(1))  # Output: 1
print(compute_factorial(7))  # Output: 5040
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                Python Debug Console

PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-pyth
.0-win32-x64\bundled\libs\debugpy\launcher' '60726' '--' 'D:\AI\AI-3.1.py'
120
1
1
5040
PS D:\AI>
```

Justification :

One-shot prompting provides the AI with one example along with the task description. The given example (Input: 5 → Output: 120) helps the model clearly understand the expected result and logic. Compared to zero-shot prompting, this approach leads to more accurate implementations with correct looping or recursion. The presence of a single example reduces ambiguity and improves correctness, showing that even minimal guidance can significantly enhance the quality of AI-generated code.

Task 3: Few-Shot Prompting (Armstrong Number Check)

PROMPT :

Generating a Python function to check whether a given number is an Armstrong number.

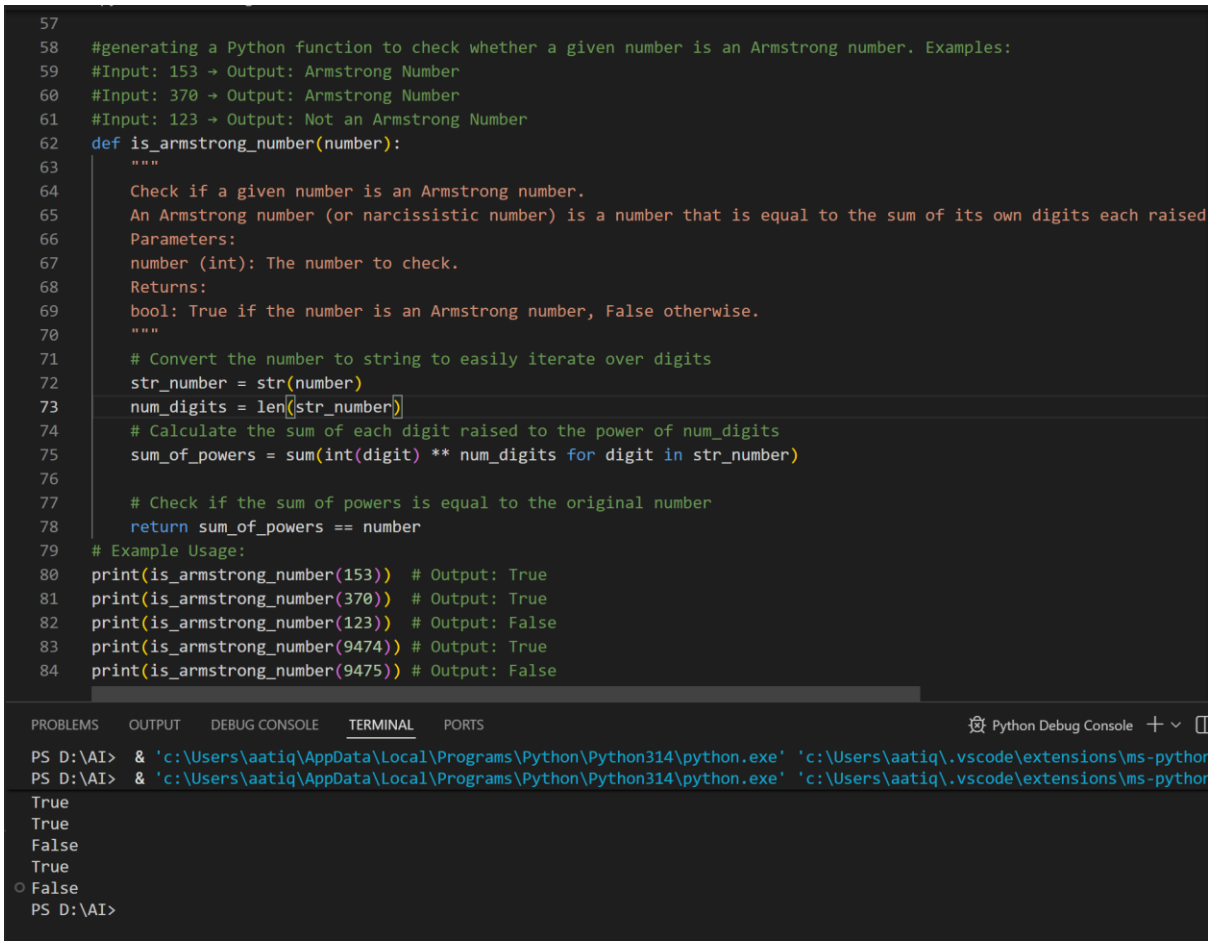
Examples:

• Input: 153 → Output: Armstrong Number

• Input: 370 → Output: Armstrong Number

• Input: 123 → Output: Not an Armstrong Number

CODE and OUTPUT :



```
#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
def is_armstrong_number(number):
    """
    Check if a given number is an Armstrong number.
    An Armstrong number (or narcissistic number) is a number that is equal to the sum of its own digits each raised
    Parameters:
    number (int): The number to check.
    Returns:
    bool: True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to string to easily iterate over digits
    str_number = str(number)
    num_digits = len(str_number)
    # 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_number)

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == number
# Example Usage:
print(is_armstrong_number(153))  # Output: True
print(is_armstrong_number(370))  # Output: True
print(is_armstrong_number(123))  # Output: False
print(is_armstrong_number(9474)) # Output: True
print(is_armstrong_number(9475)) # Output: False
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python Debug Console
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python
True
True
False
True
False
PS D:\AI>
```

Justification :

Few-shot prompting includes multiple examples, which strongly influences how the AI interprets the problem. By seeing both Armstrong and non-Armstrong cases, the model better understands digit extraction, power calculations, and decision-making logic. This results in cleaner, more structured, and mathematically accurate code. Additionally, the AI performs better when handling different test cases. Overall, providing several examples improves reliability and reduces logical mistakes, making the solution more suitable for real-world use.

Task 4: Context-Managed Prompting (Optimized Number Classification)

PROMPT :

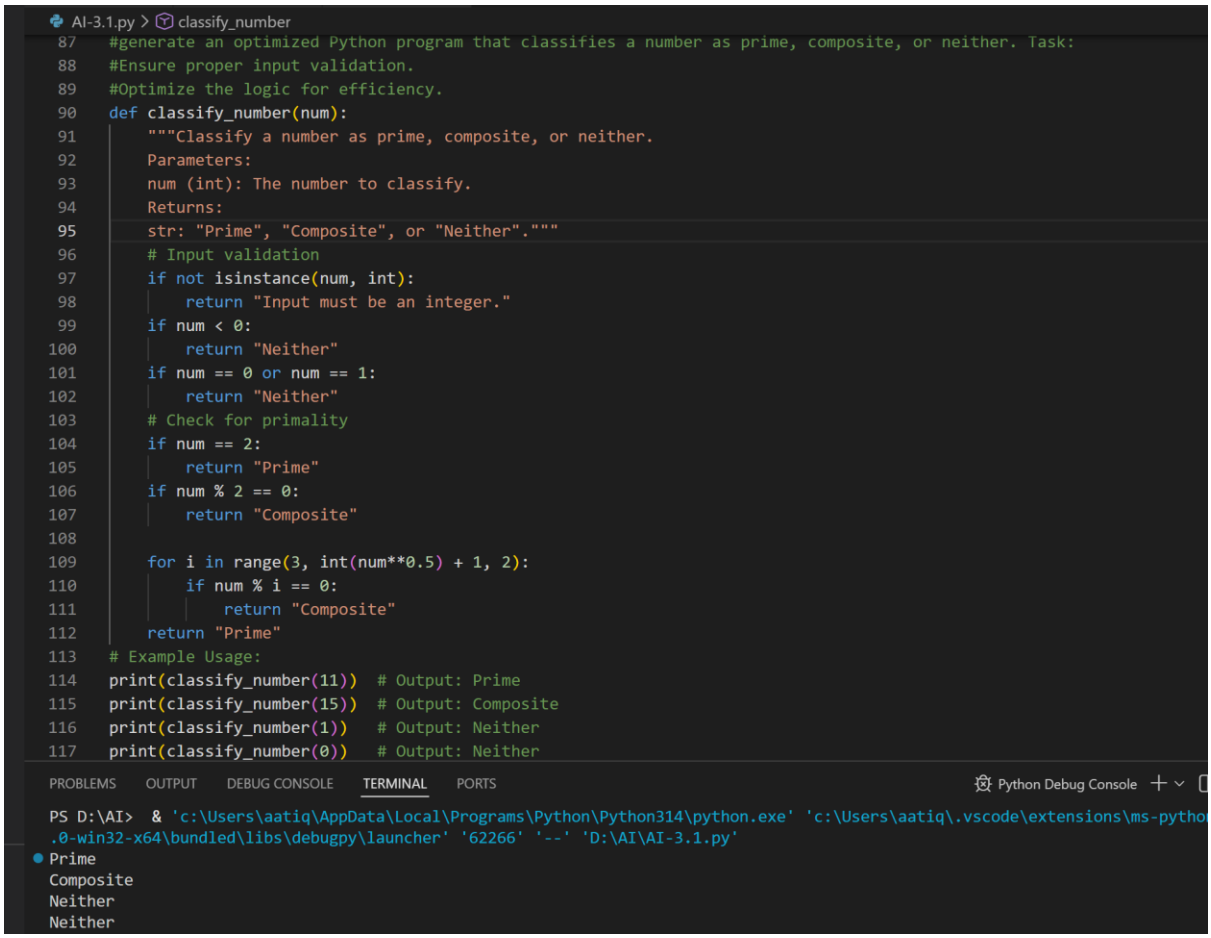
Generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

CODE and OUTPUT :



Justification :

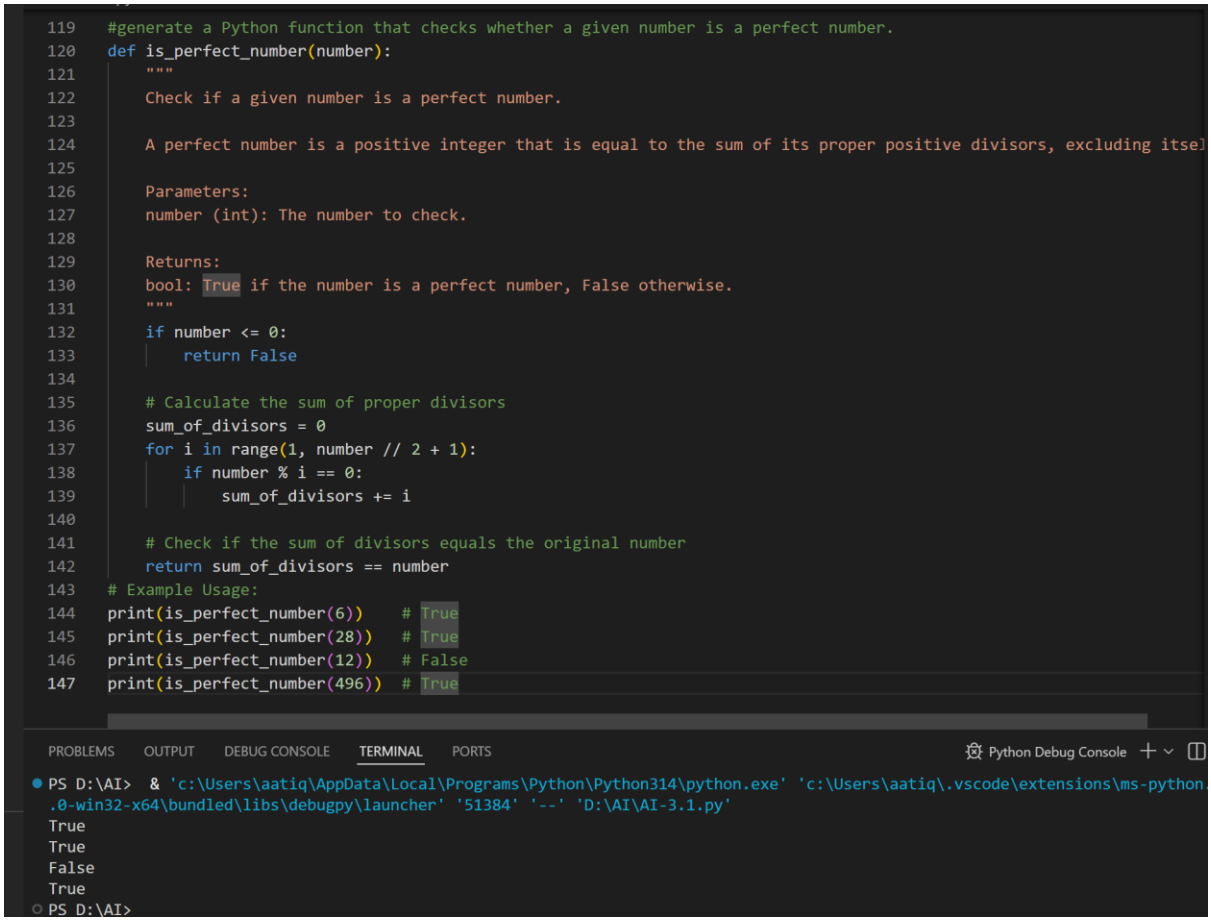
Context-managed prompting supplies detailed instructions, constraints, and performance expectations. By specifying input validation, efficiency requirements, and classification rules, the AI produces a well-optimized and complete program. The generated solution correctly handles special cases such as 0, 1, and negative numbers, and applies efficient logic by checking divisibility only up to √n. Compared to simpler prompting techniques, this method results in professional-quality, optimized, and reliable code, highlighting the importance of clear context for complex tasks.

Task 5: Zero-Shot Prompting (Perfect Number Check)

PROMPT :

Generate a Python function that checks whether a given number is a perfect number.

CODE and OUTPUT :



```python
#generate a Python function that checks whether a given number is a perfect number.
def is_perfect_number(number):
    """
    Check if a given number is a perfect number.

    A perfect number is a positive integer that is equal to the sum of its proper positive divisors, excluding itself.

    Parameters:
    number (int): The number to check.

    Returns:
    bool: True if the number is a perfect number, False otherwise.
    """
    if number <= 0:
        return False

    # Calculate the sum of proper divisors
    sum_of_divisors = 0
    for i in range(1, number // 2 + 1):
        if number % i == 0:
            sum_of_divisors += i

    # Check if the sum of divisors equals the original number
    return sum_of_divisors == number
# Example Usage:
print(is_perfect_number(6))    # True
print(is_perfect_number(28))   # True
print(is_perfect_number(12))   # False
print(is_perfect_number(496))  # True
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python Debug Console
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python.
.0-win32-x64\bundled\libs\debugpy\launcher' '51384' '--' 'D:\AI\AI-3.1.py'
True
True
False
True
PS D:\AI>
```

Justification :

In this zero-shot scenario, the AI depends entirely on its prior knowledge to generate the solution. The code usually identifies divisors and calculates their sum correctly. However, it may include inefficiencies such as checking all numbers up to n or failing to validate non-positive inputs. This demonstrates that while zero-shot prompting can generate functional code, it often lacks optimization and thorough validation. Therefore, it is better suited for basic demonstrations rather than efficient or production-level solutions.

Task 6: Few-Shot Prompting (Even or Odd Classification with Validation)

PROMPT :

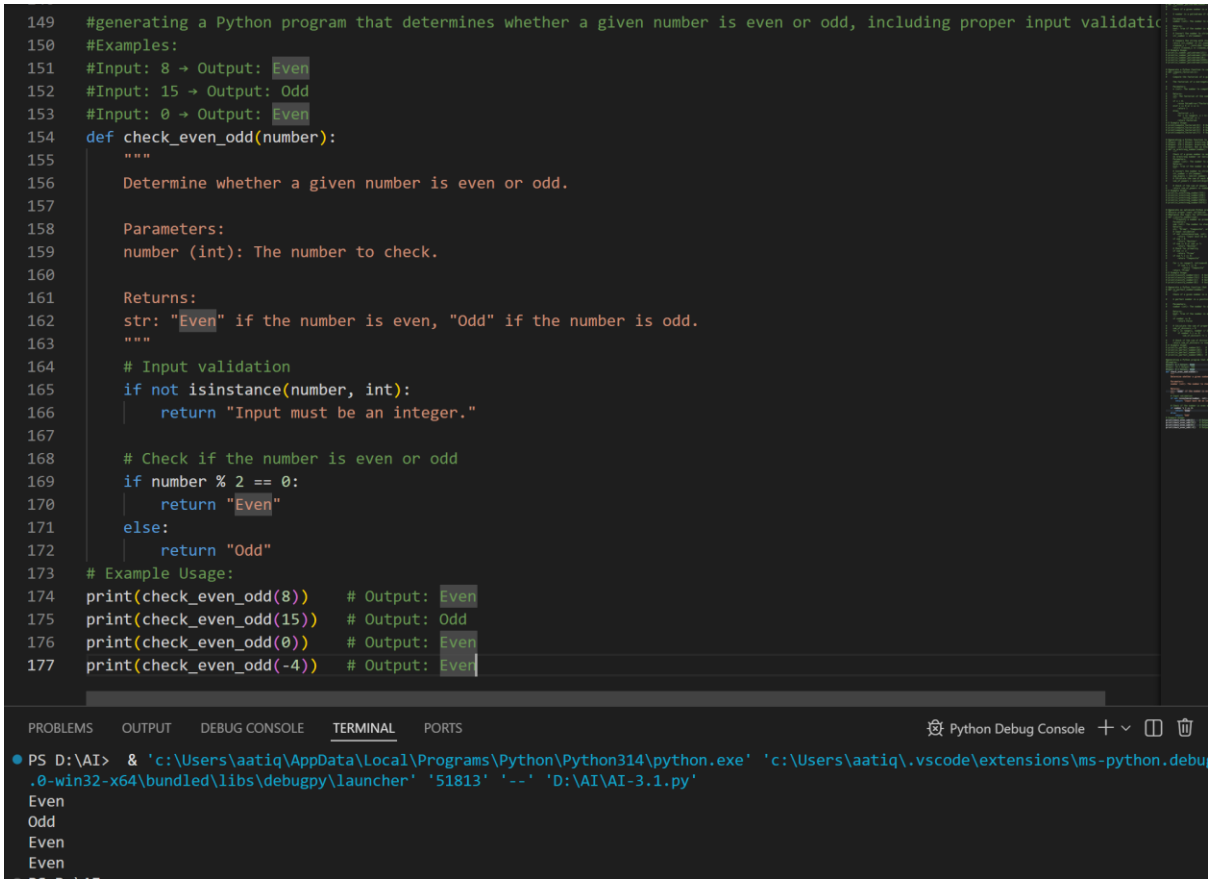
Generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd

• Input: 0 → Output: Even

CODE and OUTPUT :



```
#generating a Python program that determines whether a given number is even or odd, including proper input validatic
#Examples:
#Input: 8 → Output: Even
#Input: 15 → Output: Odd
#Input: 0 → Output: Even
def check_even_odd(number):
    """
    Determine whether a given number is even or odd.

    Parameters:
    number (int): The number to check.

    Returns:
    str: "Even" if the number is even, "Odd" if the number is odd.
    """
    # Input validation
    if not isinstance(number, int):
        return "Input must be an integer."

    # Check if the number is even or odd
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Example Usage:
print(check_even_odd(8))    # Output: Even
print(check_even_odd(15))   # Output: Odd
print(check_even_odd(0))    # Output: Even
print(check_even_odd(-4))   # Output: Even
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python Debug Console
PS D:\AI>  & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms-python.debu
.0-win32-x64\bundled\libs\debugpy\launcher' '51813' '--' 'D:\AI\AI-3.1.py'
Even
Odd
Even
Even
```

Justification :

Few-shot prompting, supported by multiple examples including edge cases like zero, helps the AI clearly understand expected behavior and output formatting. The generated program typically includes proper conditional checks, clear messages, and better handling of negative numbers. Compared to zero-shot approaches, the solution also performs better when dealing with invalid or unexpected inputs. This confirms that providing multiple examples improves robustness, accuracy, and clarity in AI-generated programs.